INSTITUT FÜR
DATENTECHNIK UND
KOMMUNIKATIONS-
NETZE

Technische
Universität
Braunschweig

Dissertation

Achieving Performance in Networks-On-Chip for Real-Time Systems

Adam Kostrzewa

Achieving Performance in Networks-On-Chip for Real-Time Systems

Dissertation an der Technischen Universität Braunschweig, Fakultät für Elektrotechnik, Informationstechnik, Physik

Achieving Performance in Networks-On-Chip for Real-Time Systems

Von der Fakultät für Elektrotechnik, Informationstechnik, Physik
der Technischen Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung des Grades eines Doktors

der Ingenieurwissenschaften (Dr.-Ing.)

genehmigte Dissertation

von Adam Kostrzewa

aus Warschau

Eingereicht am: 16.05.2018

Mündliche Prüfung am: 13.08.2018

1. Referent: Prof. Dr.-Ing. Rolf Ernst
2. Referent: Prof. Dr.-Ing. Mladen Berekovic

Druckjahr: 2018

Abstract

In many new applications, such as in automatic driving, high performance requirements have reached safety critical real-time systems. Consequently, Networks-on-Chip (NoCs) must efficiently host new sets of highly dynamic workloads e.g. high resolution sensor fusion and data processing, autonomous decision's making combined with machine- learning.

The static platform management, as used in current safety critical systems, is no more sufficient to provide the needed level of service. A dynamic platform management could meet the challenge, but it usually suffers from a lack of predictability and the simplicity necessary for certification of safety and real-time properties.

In this work, we propose a novel, global and dynamic arbitration for NoCs with real-time QoS requirements. The scheme follows design principles of Software Defined Networks (SDN) and adjusts them for the purposes of NoCs in real-time, embedded systems. The mechanism decouples the admission control from arbitration in routers thereby simplifying a dynamic adaptation and real-time analysis. Consequently, the proposed solution allows the deployment of a sophisticated contract-based QoS provisioning without introducing complicated and hard to maintain schemes, known from the frequently applied static arbiters.

The presented work introduces an overlay network to synchronize transmissions using arbitration units called Resource Managers (RMs), which allows global and work-conserving scheduling. The description of resource allocation strategies is supplemented by protocol design and verification methodology bringing adaptive control to NoC communication in setups with different QoS requirements and traffic classes. For doing that, a formal worst-case timing analysis for the mechanism has been proposed which demonstrates that this solution not only exposes higher performance in simulation but, even more importantly, consistently reaches smaller formally guaranteed worst-case latencies than other strategies for realistic levels of system's utilization.

The approach is not limited to a specific network architecture or topology as the mechanism does not require modifications of routers and therefore can be used

together with the majority of existing manycore systems. Indeed, the evaluation followed using the generic performance optimized router designs, as well as two systems-on-chip focused on real-time deployments. The results confirmed that the proposed approach proves to exhibit significantly higher average performance in simulation and execution.

Zusammenfassung

In vielen neuen sicherheitskritische Anwendungen, wie z.B. dem automatisierten Fahren, werden große Anforderungen an die Leistung von Echtzeitsysteme gestellt. Daher müssen Networks-on-Chip (NoCs) neue, hochdynamische Workloads wie z.B. hochauflösende Sensorfusion und Datenverarbeitung oder autonome Entscheidungsfindung kombiniert mit maschineller Lernen, effizient auf einem System unterbringen. Die Steuerung der zugrunde liegenden NoC-Architektur, muss die Systemsicherheit vor Fehlern, resultierend aus dem dynamischen Verhalten des Systems schützen und gleichzeitig die geforderte Performance bereitstellen.

In dieser Arbeit schlagen wir eine neuartige, globale und dynamische Steuerung für NoCs mit Echtzeit QoS Anforderungen vor. Das Schema folgt den Konstruktionsprinzipien von Software Defined Networks (SDN) und entkoppelt die Zutrittskontrolle von der Arbitrierung in Routern. Hierdurch wird eine dynamische Anpassung ermöglicht und die Echtzeitanalyse vereinfacht. Der Einsatz einer ausgefeilten vertragsbasierten Ressourcen-Zuweisung wird so ermöglicht, ohne komplexe und schwer wartbare Mechanismen, welche bereits aus dem statischen Plattformmanagement bekannt sind einzuführen.

Diese Arbeit stellt ein übergelagertes Netzwerk vor, welches Übertragungen mit Hilfe von Arbitrierungseinheiten, den so genannten Resource Managern (RMs), synchronisiert. Dieses überlagerte Netzwerk ermöglicht eine globale und lasterhaltende Steuerung. Die Beschreibung verschiedener Ressourcenzuweisungstrategien wird ergänzt durch ein Protokolldesign und Methoden zur Verifikation der adaptiven NoC Steuerung mit unterschiedlichen QoS Anforderungen und Verkehrsklassen. Hierfür wird eine formale Worst Case Timing Analyse präsentiert, welche das vorgestellte Verfahren abbildet. Die Resultate bestätitgen, dass die präsentierte Lösung nicht nur eine höhere Performance in der Simulation bietet, sondern auch formal kleinere Worst-Case Latenzen für realistische Systemauslastungen als andere Strategien garantiert.

Der vorgestellte Ansatz ist nicht auf eine bestimmte Netzwerkarchitektur oder Topologie beschränkt, da der Mechanismus keine Änderungen an den unterlie-

genden Routern erfordert und kann daher zusammen mit bestehenden Manycore-Systemen eingesetzt werden. Die Evaluierung erfolgte auf Basis eines leistungsoptimierten Router-Designs sowie zwei auf Echtzeit-Anwendungen fokusierten Plattformen. Die Ergebnisse bestätigten, dass der vorgeschlagene Ansatz im Durchschnitt eine deutlich höhere Leistung in der Simulation und Ausführung liefert.

Contents

Chapter 1: Introduction

Multi-Processor Systems-on-Chip (MPSoCs) enable high performance through integration and concurrent execution of previously separated applications and functions. In this manner, MPSoCs offer extensive sharing of intellectual property (IP) components and other system resources at low power and competitive cost [69]. Driven by their commercial success and continually increasing requirements of contemporary workloads, designers are constantly increasing the size and complexity of such architectures. Indeed, currently available MPSoCs embrace hundreds of IP cores e.g. 50 cores in Intels Knights Core [1], 256 cores in MPPA [71], 100 processing cores in Tilera's TILE-Gx line of processors [2], 1024 cores in KiloNoC architecture [57] include DSPs, CPUs and memory blocks. Further complexity growth of on-chip systems is expected to happen in the near future [25].

Such highly complex designs define a new set of requirements for the on-chip communication. The interconnect should be flexible, modular and its capacity must scale along with the number of integrated IPs. Consequently, several trends have emerged showing clear limitations of the bus-based and crossbar network architectures. Such simple interconnects although straightforward in implementation, do not scale up well along with the increasing number of connected senders [17]. For instance, in on-chip buses, the same transmission medium (i.e. link formed by a set of wires) is shared by all senders and only one node at a time may conduct its transfer which is protected by an arbiter module. Consequently, buses can serve only a limited number of IPs as their performance degrades due to serialization of all inter-IP communication which is becoming a bottleneck in System-on-Chip (SoC) architectures. As a result, the applicability of buses in complex SoCs is severely limited by the parasitic capacitance of the wires and the complexity of the arbitration [17]. Cross-bar interconnects offer an alternative solution where each sender-receiver pair has an own independent communication line. These additional lines increase aggregate bandwidth (as different transactions may proceed simultaneously) which is not possible in case of a bus. However, the higher number of long wires, which is increasing quadratically with the amount of connected

senders [62], leads inevitably to difficult timing closures of very-large-scale integration (VLSI) designs, and increases the complexity of manufacturing processes e.g. multiple synchronous regions and globally asynchronous locally synchronous (GALS) circuits. Consequently, the integration of components in a SoC is difficult, increasing the costs and complexity or even leading to unfulfilled design requirements due to the aforementioned problems at physical and logical design levels.

These challenges have raised the need for a cheaper and more flexible solution for the on-chip communication whenever the number of IP cores (communicating nodes) scales beyond the point supportable by the implementation technology for a given SoC. Consequently, at the beginning of 2000s, Networks-on-Chip (NoCs) were proposed as a viable alternative to fulfill these goals [16].

NoCs are bringing the principles of *switch*-based networks known from off-chip communication into the MPSoCs [62], [37]. In NoC-based architectures, the chip can be divided into on-chip computational units called nodes as presented in Figure 1.1. Each node is built with tiles, a router (sometimes also referred to as switch) and network interfaces. Each tile may encompass multiple hardware IPs, e.g. processors, memories or controllers or even be equipped with its own interconnect e.g. bus or crossbar. Network Interfaces (NIs) or Network Interface Units (NIUs) are pieces of equipment used for enabling communication between tiles and routers. They usually provide a common transmission protocol for tiles that use different protocols (e.g. AXI, AHB, APB, OCP, PIF or BVCI) and data formatting standards. This transmission protocol can be used instead of the tiles' own protocols or may be applied to convert the specific tile protocol to a common one. Consequently, a NI often encompass the bridge functionality. Each router is connected through a set of links with a subset of tiles as well as a subset of other nodes. Routers do not perform information processing but instead forward the data to other nodes and other processing elements called tiles, which are connected to these nodes. Links are simply sets of wires used to forward the data signals. They can be bi-directional or uni-directional depending on the implementation. The devices responsible for connecting the wires to the routers are called ports. Ports are often equipped with buffers which hold packets during times of congestion. Frequently, designers distinguish between input ports for incoming packets and output ports for outgoing packets. Buffers may be used, appropriately for input- or output-buffering or both. The node degree defines the number of links (communication channels) with other neighbouring nodes.

The communication in a network will be explained in the scope of a layer approach known from general purpose networking. Consequently, a NoC can be decomposed into a set of mechanisms (layers), see Figure 1.2. The construction of each layer denotes an independent set of mechanisms responsible for different

technical problems which can be handled independently. The transaction layer defines the communication primitives for the IP blocks belonging to tiles. It decides about the communication participants (i.e. sender and receiver or master and slave) and its purpose, e.g. memory load or store between master NI and slave NI, but not about the method use for the data transport. Consequently, the transaction layer is frequently implemented through a set of mechanisms in NI which function like bridges between different external protocols of tiles connected to the NoC (e.g. AHB, AXI), making the NoC communication transparent to its participants. The transport layer is responsible for how data are forwarded and switched throughout the interconnect. It deals with problems of packetization and addressing, but may also provide additional reliability functions as parity bits. The important part of this problem is the arbitration between concurrent packets progressing through the network performed in each router. The selected scheduling method has a great impact on average and worst-case performance metrics of the interconnect. Finally, the physical layer is responsible for how packets are converted into signals which are physically transmitted throughout the interconnect. The presented approach follows the "route packets, not wires" principle from [38]. Communication between network nodes is carried out through streams of packets on a specific route making them similar to large-scale computer interconnects e.g Ethernet.

Such designed, NoCs have numerous advantages when applied to complex MPSoCs and when compared to other simpler solutions for on-chip communication. First of all, NoCs scale significantly better. By adding a new node into the system, the raw aggregated communication bandwidth also increases. Additionally, nodes operate independently and therefore conduct a high number of transmissions in parallel. Decoupling the transaction layer from the transportation layer introduces high flexibility in a design of a network architecture for supreme performance. For example, tiles must not be homogeneous anymore. They can connect a plethora of different IP blocks using various transaction standards e.g. AXI, AHB through adjustments in NIs. This makes NoC communication "transparent" to the legacy software and components allowing backward compatibility and easy incorporation within existing MPSoCs. Also NoCs support different topologies and setups. For example, connections between nodes can be homogeneous and regular as presented in Figure 1.1, which means that each router has the same architecture, equal number of nodes and degree. However, NoCs may also be heterogeneous and irregular. In such setups, routers can operate with various frequencies, support different number of ports and even different width of links between nodes (i.e. different bandwidth) as well as different packet scheduling techniques. This allows NoCs to span across MPSoCs with various IP blocks, independent clock domains and available chip space. Finally, application of switches improves control on the wires

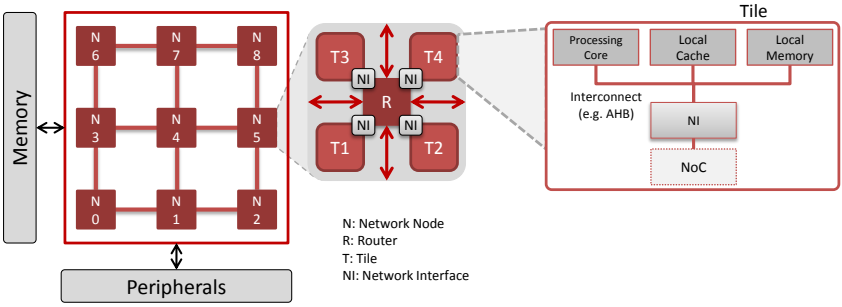


Figure 1.1: Example of a NoC-based MPSoCs platform composed of Network Nodes and Tiles.

which are becoming shorter. Therefore, switches simplify timing closure and reduce overall cost of the system including power consumption and production efforts. By applying NoCs in a complex system, the designer has a fine-grained control on the communication properties allowing performant, flexible and resource efficient systems on a chip.

The aforementioned features of contemporary NoC architectures have contributed to their wide commercial success in the domain of the multi-functional consumer and mobile devices. Therefore, as the complexity of setups in the safety-critical embedded domains (e.g. automotive, avionic, industrial robotics) increased significantly in recent years, NoCs have gained researchers' and designers' attention as a new key-enabling technology.

1.1 New Challenges in Design of Safety-Critical Systems

In many embedded applications high performance requirements have reached safety critical real-time systems. For instance, an average, modern vehicle has approximately 25 - 30 Electronic Control Unit (ECU) whereas some high-end models are have over 100 ECUs, see Figure1.3. The combined contemporary in-vehicle network with the wiring is the second heaviest component in the vehicle (behind the engine), having over 6 km of copper wire and weighing over 70 kg [109]. These numbers are foreseen to rise in the nearest future with an advent of advanced features, e.g. autonomous driving.

ECUs are also becoming more complex and highly interconnected for new levels of comfort, safety and efficiency. Fig 1.4 presents examples of such features in a car

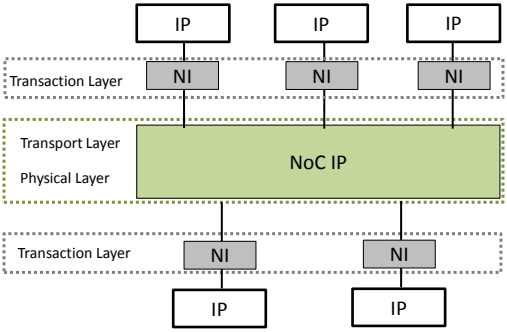


Figure 1.2: Layers of the NoC-based communication in the MPSoC.

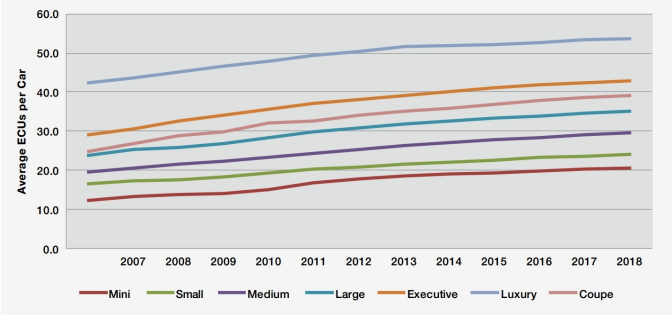


Figure 1.3: Number of ECUs in different segments of automotive production, source [109].

e.g.: camera-based control (lane departure warning, automatic cruise control, etc.) sensors and actuators used for measurements in mechanic control applications (motor, brakes) e.g. gase (COx, NOx, etc.), temperature , vibration, wheel speed, torque, yaw etc.

Besides these features, upcoming highly automated and autonomous driving introduce a new set of requirements, see Figure 1.4 (red boxes). These include e.g. robust and efficient surround sensing and decision making including machine learning and control. The system must offer high performance and parallel processing (e.g. big data crunching, decision making with convolutional neural networks) as well as synchronization of different ECUs over heterogeneous interconnects e.g. Ethernet switches and buses. The sensor fusion is done in real-time

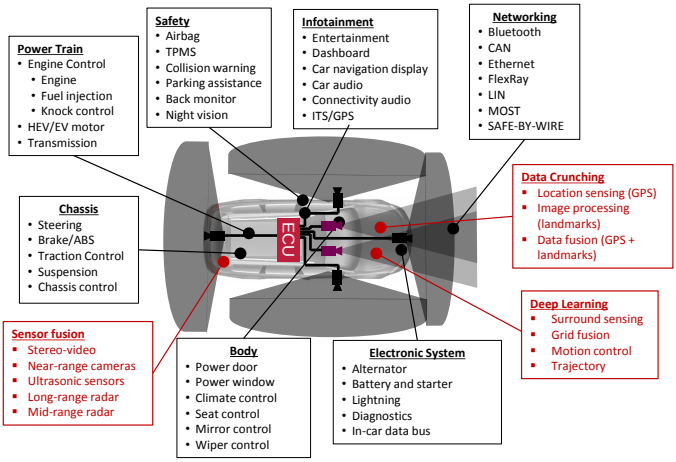


Figure 1.4: Overview of the electronic features in a modern vehicle including ADAS.

while gathering and processing data from plethora of sensor inputs (i.e. high resolution sensor fusion). Actions must be taken in the context of a dynamic, constantly changing environment (e.g. situation on the road, quality of sensor data) as well as platform pitfalls (e.g. transient overloads, memory overflows, data loss and missed deadlines). Finally, these systems have strict design constraints with respect to limiting power consumption and cost budget.

Consequently, to keep up with these high demands, multi- and MPSoCs have become architectures of choice also in the safety critical domains. In case of automotive designs this has three significant benefits: aggregation, extension and redundancy. Aggregation property refers to design schemes which decrease the number of ECUs by moving the software from multiple smaller ECUs (e.g. low utilization) into one many-core ECU (e.g. higher utilization and thus lower cost). Extension property permits integration of higher number of features in the same ECU e.g. new advanced functionalities such as ADAS or parallelization of the code execution. Finally, the redundancy property ensures the combination of high performance with high reliability on the same chip. Therefore, some ECUs or functions may be duplicated (e.g. executed on different cores) to compare their results and look for erroneous behavior.

As an inevitable consequence of integration, handling these new workloads requires flexible on-chip networks which capacities would scale along with the size

and multiplicity of features in the design.

The advantages of NoCs described in the previous section, which greatly contributed to the commercial success of SoCs, MPSoCs and MSoC in the domain of general purpose computing, raised designers' attention for their application also in the embedded domains. However, soon it has become clear that their implementation will not be as straightforward as it might have seemed due to the new requirements resulting from the deployed workloads and physical environments in which they are applied.

Firstly, many of the workloads in the embedded domain have real-time constraints which require not only the correct calculation/operation results but also timely responses. Secondly, some of the electronic equipment can be critical to the mission or user safety i.e. its malfunction could cause serious harm to the user or system within which it is integrated. Consequently, these devices must be designed according to the safety standards and must frequently undergo the rigorous certification process before their market deployment. This requires from that producers assure methods for the integration, testing and verification of their products.

These requirements fully apply to the automotive domain and are especially important to the new advanced subsystems such as ADAS. It is easy to predict that a delayed signal from a sensor (camera or radar) can have disastrous effects on the vehicle and its surroundings. Moreover, integration in the same SoC of multiple different features could delay processing of the sensor data and result in similar fatal consequences due to the accesses to shared resources [107].

In Sections 1.2 and 1.3 these new aspects of the SoC design are discussed in greater detail with focus on real-time and safety. The dissemination goal is to provide a brief overview before detailed discussion on how these challenges translate into the NoC requirements, which follows in Section 1.4. Although the presented discussion focuses on the automotive domain many of its aspects can be directly extrapolated on other embedded domains such as industrial automatics, avionic or robotics. Therefore, whenever possible these common challenges and solutions will be highlighted (e.g. standardization processes) for the sake of better understanding.

1.2 Real-Time Applications

It is a common intuition that many automotive functions, especially when controlled with ECUs, have temporal constraints , e.g., breaking systems, steering or engine control. The upcoming ADAS functionality extends this spectrum even fur-

ther by gathering sensor data (e.g. timely video frames and / or radar signals), its processing (e.g. denoising, filtering, compression), and decision making (e.g. big data processing and machine learning with convolutional neural networks). However, real time requirements appear also in other electronic domains, e.g., traffic control, multimedia, mobile communication, medical applications, industrial robotics or avionic. In this section, we provide a brief overview and summary of the most important properties of such setups with a special focus on their requirements towards the communication architecture.

1.2.1 Real-Time Traffic Requirements

In systems with real-time (temporal) requirements, the correctness of the system design and working depends equally on temporal and functional aspects. Real-time applications/systems must guarantee not only that their results are logically correct, but also that they are delivered on time. The physical time by which a specific result must be produced is called deadline. Deadlines are frequently dictated by the environment and physical nature of the controlled process.

Fundamentally, real-time applications can be classified by a metric, which uses the consequences that a deadline miss can cause. The theory of real-time systems design [92] distinguishes three application categories while considering the aforementioned criterion: hard real-time (HRT), soft real-time (SRT) and best-effort (BE). These categories will be briefly discussed, as properties of the applications directly influence the properties of the initiated NoC traffic, i.e., a hard real-time application has hard real-time communication requirements with respect to bandwidth or latency. However, the presented classification is orthogonal to the different traffic metrics described in the next sections. For instance, it is possible to have a hard-real time application requiring guaranteed worst-case latencies or a soft-real time application requiring guaranteed throughput.

HRT applications have firm deadlines, i.e., the utility of the produced result is zero when the deadline is crossed. Consequently, any delay in their execution, including high latencies of initiated transmissions, may have severe consequences for the whole system, i.e., fatal faults and prohibitive degradation of service. Note, that this frequently includes situations when the user's safety may be in danger, e.g., activation of anti-lock breaking system (ABS), high latency of video frames from cameras in an ADAS system leading to malfunction.

Indeed, as presented in Figure 1.5, in practice many safety critical systems have hard deadlines - hence systems are both safety and time critical. Therefore, transmissions conducted by HRT senders, i.e. hard real-time transmissions (HRTTs) are usually not allowed to miss any deadline, i.e., its worst-case latency must stay

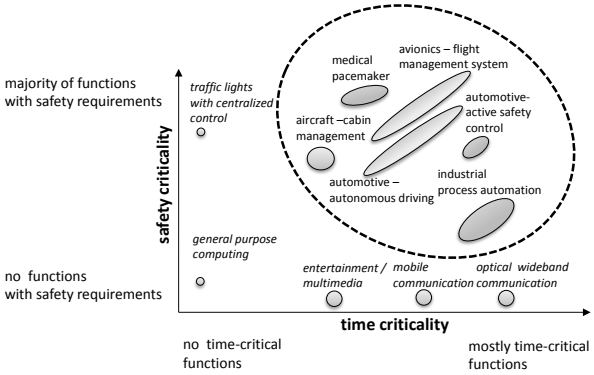


Figure 1.5: Dependency between safety and time-critical systems, based on [50].

within assumed upper/lower limit. Similarly, even if the functionality does not directly affect user safety it may have hard real-time requirements from the producers point of view, as a failure of the service could cause client loss or a substantial financial penalty. Accommodation of the traffic from HRT applications requires worst-case dimensioning during the design process and, in case of safety-critical systems, also a verification proving adherence to the standards. Due to these requirements the characteristics of traffic originated from hard real-time senders is usually well specified and tested, e.g., periodic DMA transfers.

Similarly to HRTTs, transmissions initiated by SRT senders, i.e. soft real-time transimssions (SRTTs), must also comply with the overall real-time performance objectives, e.g., guaranteed latency or throughput. The main difference, when compared to hard real-time senders, is that these applications are rarely required to rigorously meet all their deadlines, i.e., the produced results have some utility after the deadline, see Figure 1.6. For instance, video streaming done as a part of infotainment functions in a car or a plane does not influence vehicle safety, but video frames must still arrive with a certain latency to prevent quality drops and glitches. Another example are control algorithms based on a feedback loop. Frequently, the algorithm may tolerate a limited number of cases when instead of sampling new data old values are used e.g. [156, 143, 125]. However, also in case of SRT senders timing can be a critical factor depending on other non-functional requirements. For a producer of an infotainment system the quality of its users' experience may play a critical role in the market success of a product. Consequently, it may accept a sporadic drop of the video quality but may lose clients whenever it happens too

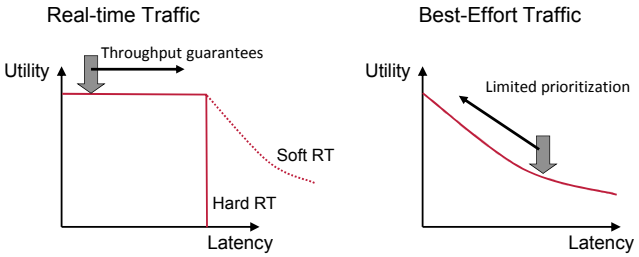


Figure 1.6: Communication requirements of a NoC traffic concerning the temporal isolation, based on [132].

often.

The last category of best-effort (BE) senders is populated by the majority of existing applications. In contrast to both aforementioned real-time classes, BE transmissions have no strict temporal requirements, i.e, a deadline missed by them does not endanger the system. Consequently, the behavior of best-effort applications is rarely tested with respect to temporal properties, e.g., the number, duration and frequency of memory accesses. BE senders are usually designed targeting average performance metrics, such as average latency, average throughput and compiled with standard toolchains without considering the temporal properties nor certification requirements, e.g., GNU compilers and applications running on processors with caches. Consequently, in case of initiated accesses to the on-chip interconnect BE senders may exhibit high burstiness, i.e., an unpredictable and highly variable resource usage (number of and length of transmissions). BE senders suffer from a lack of temporal models and their integration with other applications having real-time requirements is difficult.

Figure 1.6 presents a summary of temporal properties of NoC traffic. The X-axis depicts worst-case communication latency and the Y-axis the utility of data, i.e., the usefulness of data for the receiver. The utility value equal to zero denotes a missed deadline. The timely arrival of data (before or on deadline) is denoted by the utility value equal to one. All other values of the utility function (between these bounds) denote a performance degradation. For instance, HRTTs have utility only if they arrive before the deadline. For SRTTs utility function (usefulness of the data) decreases after the missed deadline.

1.2.2 Integration of Traffic

As discussed in Section 1, NoCs are foreseen as a communication backbone for large SoCs integrating different ECUs. As a result of such integration, it can happen that diverse traffic classes, i.e., HRTTs, SRTTs and BE, must share the SoC resources. This integration causes co-dependencies between applications running on different cores, which may endanger safety. Unpredictable and bursty accesses from BE senders may lead to contention in network buffers.

In on-chip interconnects without appropriate quality-of-service (QoS) mechanisms, resources are not reserved in advance, i.e., transmissions are scheduled as soon as they arrive, and all traffic receive equal treatment. Because of that, some interference from BE traffic may lead to missed deadlines by SRTTs or HRTTs.

On the other hand, there is often no advantage in completing an HRTT earlier than required. The design process (e.g., safety standards) usually insist on the satisfaction of all timing constraints even if unnecessarily stringent. Therefore, as long as an application completes by its deadline, its response time is not important [132, 144]. This property could be used to slow-down SRTTs and HRTTs for accommodating BE traffic. Finally, the majority of best-effort applications, although not timing critical, is still latency-sensitive. Their performance degrades with higher latency as presented in Figure 1.6. For instance, applications running on processors with caches must frequently fulfill temporal requirements to profit from low average-case latencies for improved processor utilization. Consequently, they profit from higher interconnect performance and higher SoC resource share. These properties lead to contradictory requirements whenever different classes of real-time traffic share the interconnect.

The integration of HRTTs, with SRTTs and BEs in the same chip, requires enforcing *sufficient independence* between components, i.e., assuring that the behavior of HRT is independent of the behavior of SRT and BE senders. However, this often compromises the performance of SRTTs or BEs. Quality-of-Service mechanisms for on-chip interconnects usually prioritize hard real-time senders over best-effort traffic and soft real-time traffic. Hence, BE traffic suffers from high latency although time-critical traffic has no to little benefit from reduced latency.

Moreover, worst-case dimensioning required for the deployment of HRT senders leads frequently to resource overprovisioning. However, during regular work of the system, such extreme conditions may rarely occur and such arbitration results in a significant drop in average utilization, i.e., underutilized resources. Consequently, an efficient co-execution of such diverse application types is still an open research question with possibly high engineering and economic impact, i.e., is critical to the success of NoCs on the market.

1.3 Safety Standards and Certification

As already discussed, although not all real-time systems are safety critical, the majority of safety-critical systems have real-time constraints . Safety critical systems are system where a malfunction can cause an "unacceptable risk of physical injury or of damage to the health of people, either directly, or indirectly as a result of damage to property or to the environment" following the definition from [5]. Therefore, since several years, one may observe accumulation of industrial and research efforts towards standardization of the safety life cycle for electronic products.

This process is already at an advanced stage in the case of the avionics industry and the domain of industrial automatics. In the avionic domain, the development and validation of software is regulated by the standard DO-178b [4] and of underlying hardware components by DO-254 [3]. During a rigorous validation process the producers must prove compliance of their products to the requirements and production methods enforced by these standards, which plays a decisive role for aircraft approval by a certification authority such as the Federal Aviation Administration (FAA). A similar process is also enforced for the development of heavy machinery (IEC 62061) and system for process industries (IEC 61511), railway (IEC 62279) and power plants (IEC 61513).

In case of automotive industry, safety standards have been introduced relatively late.[1]. The major industrial efforts have led to the establishment of the ISO26262 [67] standard derived directly form the IEC 61508 [5] standard document in 2011. The IEC 61508 introduced by the International Electrotechnical Commision (IEC) proposed a generic approach for the safety life cycle of all systems comprised of electrical and/or electronic elements including programmable elements. Its main goal was to establish a foundation, which can be adjusted for different branches of the industry, e.g., rail, machinery, power plants. Therefore, the ISO26262 constitutes in many aspects a straightforward extension of this approach adjusted for the purpose of the automotive domain. However, there are also differences. For instance, the original IEC 61508 considered low volume fabrication of components, whereas ISO26262 is focused on large quantities, i.e., serial production.

According to ISO26262 safety is defined as "the absence of unreasonable risk". Solving this problem in case of electrical and electronic devices requires incorporation of the appropriate measures throughout the design process. Consequently, ISO26262 focuses and refines the V-Model [67] presented in Figure 1.7. It offers

[1]The reasons for these are: 1) consequences of a car crash are considered to be minor in comparison with an avionic accident; 2) the liability constraints enforcing certain quality from automotive manufacturers

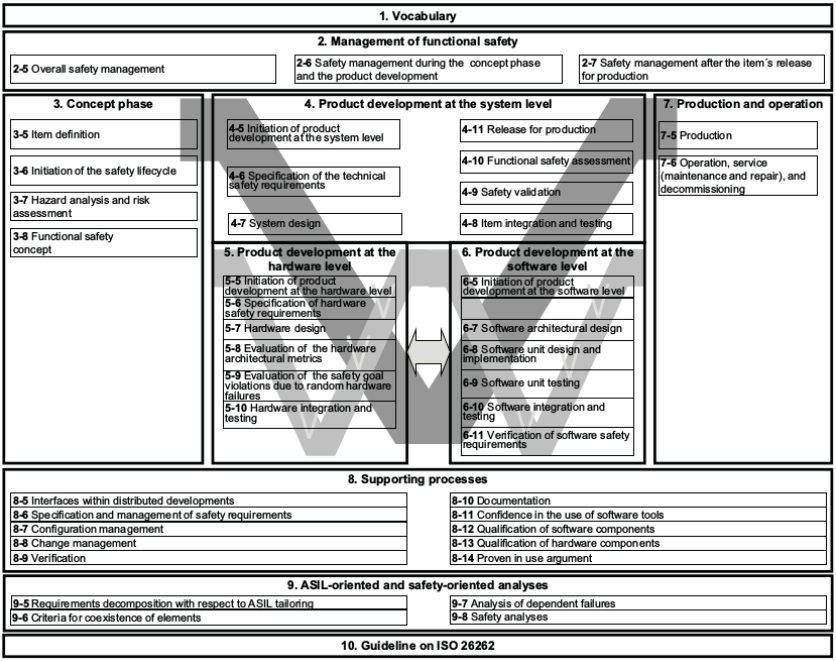


Figure 1.7: V-Model according from ISO26262, source [67]

a top-down approach towards the engineering problem, Firstly, the requirements must be delivered while considering: a) the specification of the intended functions and their interactions necessary to achieve the desired behavior of the system (functional concept) as well as b) quantification of the tolerable risk. Although the former task is straightforward the latter can be demanding. It usually requires formalized methods and processes for achieving the assurance level required by standards and certification authorities. This is done through a model driven design, where the producer must provide not only the product but also sufficient data to verify and test the design (i.e. artifact) independently.

Therefore, NoCs, whenever used for communication between safety critical IPs such as automotive functions, are or will be, depending on the safety-critical domain, the subject of regulation through standards and certification procedures to assure their correct functioning. In this context, not only the possibly high average

performance and low costs play a critical role but also, even more importantly, the ability to prove adherence to the safety requirements. This adds another complexity layer to the design process and requires traceability with respect to real-time properties, e.g., application of formal analysis methods such as Real-Time Calculus [138], Network Calculus [89] or Compositional Performance Analysis [61]. Next, details on which of the ISO26262 requirements for the data communication are applicable to Networks-on-Chip will be presented.

1.3.1 Sufficient Independence

There are multiple sources of possible hazards in a MPSoC which differ in the severity and exposure. The process of integrating several ECUs in the same system-on-chip leads to a setup in which not all integrated IPs might have the same impact on the system safety (i.e. criticality) and thus a mixed-criticality system is created. For instance, some of them might not be designed considering user safety, e.g., worst-case behavior. Consequently, their behavior cannot be trusted. From now on, these IPs (hardware or software) will be called *non-critical* and they constitute the majority of BE traffic. Moreover, even safety critical IPs may have different impact on users safety and therefore require different certification efforts. In the automotive context, ISO26262 distinguishes between four different Automotive Safety Integrity Levels (ASIL A-D) defining the relative level of risk-reduction provided by a safety function. Each of them defines more restrictive risk analysis and safety goals.

For such mixed-criticality systems, safety standards require certification of the whole system to the highest relevant safety level (e.g. highest ASIL) or temporal and spatial separation. For instance, the IEC 61508 states "If the safety integrity requirements for these safety functions differ, unless there is sufficient independence of implementation between them, the requirements applicable to the highest relevant safety integrity level shall apply to the entire E/E/PE safety-related system.". Similarly, ISO26262 states that "If the embedded software has to implement software components of different ASILs, or safety-related and non-safety-related software components, then all of the embedded software shall be treated in accordance with the highest ASIL, unless the software components meet the criteria for coexistence in accordance with ISO 26262-9:2011, Clause 6.", where Clause 6 proposes "In the case of the coexistence of sub-elements that have different ASILs assigned or the coexistence of sub-elements that have no ASIL assigned with safety-related ones, it can be beneficial to avoid raising the ASIL for some of them to the ASIL of the element. When determining the ASIL of sub-elements of an element, the rationale for freedom from interference is supported by analyses of dependent failures

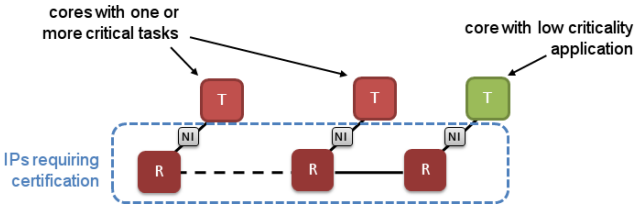


Figure 1.8: An example of a NoC in with a mixed-critical workload. The interconnect must be certified according to the traffic with the highest criticality or must provide QoS mechanisms for sufficient isolation.

focused on cascading failures". The freedom of interference is later defined as "absence of cascading failures between components that could lead to the violation of [some] safety requirements". By applying these rules to the system-on-chip, the parts of the HW and RTE, which are always used, must be certified to the highest relevant safety level. For all other components "sufficient independence" must be implemented, see Figure 1.8. As non-critical tasks cannot be trusted (e.g. unknown execution time, activation frequency, communication volume) network interfaces must separate the critical network from non-critical tiles. Additionally, routers must provide a predictable upper bound on the worst-case interference between concurrent transmissions.

1.3.2 Communication Faults and Real-Time Properties

According to ISO26262 the communication of safety-related data must be protected at run-time against effects of faults which may lead to failures of the system. These faults include transient faults, e.g. single event upsents such as bit-flips from electro-magnetic radiation leading to corrupted data, physical damage as well as lack of sufficient independence between tasks. Consequently, ISO26262 provides a list of faults, presented in Table 1.1 regarding the exchange of information which must be considered in case of an interconnect for certification purposes. In this context, the end-to-end protection defines a set of mechanisms which allow a reliable detection of these faults and appropriate countermeasures. In the NoC context, some of these faults directly refer to the real-time metric for the on-chip interconnect, e.g., a delay of information or blocking access to the communication channel. Others relate to the protection of packets done directly in routers, e.g., without a backpressure signal/ flow control (informing about full buffers in ports) packets can be overwritten leading to corruption of information.

Fault Type	Description
Repetition of information	A type of communication fault, were information is received more than once.
Loss of information	A type of communication fault, were information or parts of information are removed from a stream of transmitted information.
Delay of information	A type of communication fault, were information is received later than expected.
Insertion of information	A type of communication fault, were additional information is inserted into a stream of transmitted information.
Masquerade or incorrect addressing	A type of communication fault, were non-authentic information is accepted as authentic information by a receiver.
Incorrect sequence of information	A type of communication fault, were information is accepted from an incorrect sender or by an incorrect receiver.
Corruption of information	A type of communication fault, which changes information.
Asymmetric information from sender to multiple receivers	A type of communication fault, were receivers do receive different information from the same sender.
Information from a sender received by only a subset of the receivers	A type of communication fault, were some receivers do not receive the information.
Blocking access to a communication channel	A type of communication fault, were the access to a communication channel is blocked.

Table 1.1: Summary of communication faults which must be considered in the design of the automotive systems, from ISO26262 [67]

Note, that other faults, although not directly related to the temporal metrics, frequently influence temporal predictability indirectly. Transient errors, malfunctioning or malicious senders may introduce uncertainty and dynamics to the system, e.g., sporadic overloads due to re-transmissions or "Babbling idiots". Therefore, their contribution may significantly decrease the worst-case estimation of the system performance.

1.4 Requirements for Real-Time and/or Safety-Critical Workloads

NoC architectures are judged by four measures [37]: production cost, transmission latency, throughput and energy/power consumption. The two latter factors are performance metrics. Latency denotes the time necessary for a packet to traverse the interconnect whereas the throughput defines the amount of data per time unit which can be moved through the network (e.g. in bits per second). Consequently, the majority of NoC architectures target a possibly high *average performance* [37]. In the following, such NoCs will be addressed as common or performance optimized NoCs/architectures.

However, even if the NoC design achieves high utilization at low cost it is still difficult to guarantee system predictability in real-time domains like avionic or automotive. Whenever concurrent transmissions compete for the interconnect resources (wires, buffers) the resulting interference couples the execution of applications running on different cores. The network architecture must assure that this interference is predictable and stays within the assumed limits. Otherwise, a sender may not comply with its temporal requirements.

Therefore, providing predictable arbitration between concurrent transmissions is a critical factor for the design of NoCs in SoCs with temporal requirements. Moreover, the support for temporal properties should not cancel out benefits resulting from the application of NoCs, e.g., high efficiency, scalability, flexibility and low production costs. This requires that the designer adjusts the arbitration of interconnect resources for incorporation of dynamics resulting from system integration, e.g., simultaneously hosting different real-time and/or safety traffic classes, incorporation of system dynamics. Mechanisms designed for this purpose should be not only efficient and affordable but also provide a possibility of straightforward verification and testing as required by safety standards. The summary of these requirements is presented in Table 1.2. Next a detailed discussion and justification will be presented.

Code	Reuqirement	Metrics	Full Description
R1	Traffic Types	Real-Time Performance Safety Costs	NoC should provide support for multiple traffic classes e.g. GL, GT, BE
R2	Workload Integration	Real-Time Performance Costs	NoC should provide efficient support for real-time requirements without need to modify legacy code and other IPs
R3	Flexibility	Performance Costs	NoC architecture should be able to support different resource allocation strategies depending on particular setup
R4	Dynamics	Performance Safety Costs	NoC architecture should allow efficient and safe incorporation of dynamics in system behavior
R5	Fairness	Real-Time	NoC architecture should provide fair allocation of interconnect resources whenever RT senders have different requirements

Table 1.2: Requirements for contemporary and future NoC architectures for deployment in real-time and safety-critical domains e.g. automotive (part 1).

Code	Reuqirement	Metrics	Full Description
R6	Mixed-Criticality	Safety Performance	NoC architecture should provide as high average performance to the BE senders as possible in mixed-critical setups
R7	Switch-off QoS	Costs Energy	NoC architecture should provide a possibility to switch-off real-time and/or safety mechanisms whenever there are no senders with such requirements deployed
R8	Scalability	Costs Energy	NoC architecture should provide performance (average and worst-case) proportionally to the load at runtime (i.e. work conserving arbitration) and not other static factors e.g. number of senders
R9	Locality	Performance Real-Time Safety	NoC architecture should preserve the order of arriving packets whenever it is necessary
R10	Verification	Safety Costs	NoC architecture should allow efficient formal verification for standartization/certification purposes
R11	Detect NoC HW faults for higher ASIL levels	Real-Time Safety	Errors must be detected in order to initiate countermeasures (for fail-safe and fail-operational behavior).

Table 1.3: Requirements for contemporary and future NoC architectures for deployment in real-time and safety-critical domains e.g. automotive (part 2).

Support Different Traffic Types (R1,R5,R6)

The main purpose of a real-time NoC is to assure that temporal properties of transmissions comply with the timing requirements of the integrated applications. These requirements for real-time traffic are usually defined with the notions of worst-case latency and minimum throughput. The former defines the upper bound on the duration of the communication, e.g., maximum latency of a single packet transmission. The latter refers to the lower bound on the amount of data transferred per unit of time.

Consequently, the distinction between these two classes of safety-critical workloads is visible in many NoC designs. The transmissions are then classified as:

- *Guaranteed Latency (GL)* traffic, requiring strict guarantees for each initiated transmission, e.g., control traffic, synchronizations or interrupts. Note, that GL senders usually require low latencies but at the same time transmit short messages, i.e., low communication volume.

- *Guaranteed Throughput (GT)* traffic, requiring strict temporal guarantees per certain data/transmission volume. Consequently, the sender requires service guarantee per whole logical transmission which can be composed of several packets rather per each packet independently. For example, in case of streaming applications using DMA engines for cyclic transfers it is important that the whole chunk of data is available at a certain moment in time. Consequently, the latency of single packets may vary as long as the deadline for the burst is held.

- *Best Effort (BE)* traffic, having no strict requirements with respect to timing, i.e., no strict latency and throughput requirements are known which could endanger the system safety. However, this does not mean that BE senders have no timing requirements. BE traffic profits often benefits from lower latencies as it originates from "general-purpose" applications such as typical desktop or infotainment functionalities. These senders (by design) profit from low average temporal metrics, e.g., low average latency or high throughput, since low latencies increase the utilization of cores and therefore of the whole system. Low average latencies of BE traffic can be used, for example, to improve user experience in case of infotainment systems.

These traffic classes constitute the basis for the designs. Combinations of requirements are possible, e.g., high latency, low throughput traffic. Moreover, the knowledge and understanding of the workloads may be exploited to optimize the designs for instance by safely postponing some the packets from GT transmissions

to improve performance of BEs. Consequently, the NoC architecture for real-time workloads should provide support for multiple traffic classes to ease their incorporation. This is reflected in requirement R1 "Traffic Classes" in Table 1.2.

Moreover, whenever a NoC is used to host multiple applications with different requirements this should be reflected by the interconnect arbitration. First of all, even if different transmissions have hard real-time or soft-real time requirements they must not necessarily be the same. As described in Section 1.2.2, some transmissions may finish later than others and still be on time. Therefore, the arbitration in the NoC should be fair - offer sufficient resources for the sender to comply with its requirements but avoid overprovisioning. This is reflected in requirement R5 "Fairness" in Table 1.2.

As discussed in Section 1.3.1, this is especially important in mixed-critical setups where there is usually no advantage in completing an HRTT earlier than required since safety standards insist on the satisfaction of all timing constraints even if unnecessarily stringent, i.e., as long as an application completes by its deadline, its response time is not important. In contrast, transmissions from soft-real time and best effort applications (SRTTs) are rarely required to rigorously meet their deadlines but at the same time they must still comply to the overall real-time performance objectives, e.g., low average latencies. However, many of safety mechanisms compromise performance of BEs and SRTTs which should be avoided. This problem is reflected in requirement R6 "Mixed-criticality" in Table 1.2.

Integration Efforts (R2,R8,R9)

The previous section explained why the application of NoCs for workloads in real-time and safety-critical domains is much more challenging than in the case of general-purpose computing. Note that the aforementioned temporal requirements do not exclude using other design goals in conjunction. Consequently, in the ideal case, supporting real-time properties should be transparent to other integrated application and do not decrease their performance nor require exceptional integration efforts.

Firstly, integration of real-time senders in a NoC should not need extensive modification of IPs, e.g., legacy code or design of hardware components. This requirement is not only essential due to the increased production costs, which such modifications could cause, but also because producers frequently apply proprietary components and have no rights and/or possibility to adjust them. Moreover, even slight adjustments in safety-critical domains could cause the costly process of verification and re-certification. The requirement R2 "Workload Integration" in Table 1.2 reflects these challenges.

Moreover, mechanisms for QoS should not limit the scalability of NoC architectures. Although the strict separation of transmissions through static resource allocation assures QoS guarantees, it usually results in high overhead. For example, in performance-optimized NoCs, to guarantee the safety of critical senders, mapping algorithms try to avoid overlapping of paths used by safety-critical and non-critical, best-effort (BE) traffic, i.e., spatial separation of transmissions. Otherwise, the critical streams may not be able to meet their deadlines. Such mapping typically leads to prolonged paths for BE traffic, which degrades the BE performance as these are often latency-sensitive [104]. Hence, although BE sender are rarely required to meet all the deadlines rigorously, they must still comply with overall real-time performance objectives as low average latencies. At the same time, the static mapping based on the worst-case behavior frequently leads to a significant decrease of hardware utilization. In setups with sporadic safety-critical transmissions resources (paths) reserved for them are rarely used e.g. [7]. Consequently, the performance of the NoC architectures should be influenced (in the optimal scenario) only by the workload initiated at runtime and not by static factors such as the number and type of integrated senders. The requirement R8 "Scalability" addresses this challenge.

Finally, the NoC design can additionally assure the following traffic properties [37]:

- *traffic locality* requires that: i) packets belonging to the same logical connection arrive in the same specific order in which they were injected to the NoC (in-order delivery) and / or ii) that streams from two different senders targeting the same node do not mix in the NoC. This is especially important in case of state-dependent peripherals, such as DDR-SDRAM memories, for which performance, response time and power consumption depend on the history of previous accesses [81].

- certain jitter/*varying arrival rate* (i.e. arrival rate) as some applications require certain granularity of data to proceed with execution, e.g., the rate of arrival of video frames for a video decoder. The goal is to limit the arrival jitter which could appear in case when non-bursty traffic mixes with a strongly varying in time.

In many deployment scenarios, protecting locality of connections increases the utilization of the peripherals and IP components. This happens, for example, with latencies of the DDR SDRAM memories. Indeed, SDRAMs have an internal level of caching, which is equal to 8kB in the standard DDR3 modules. Consequently,

contiguous and aligned 8kB long transfers fully benefit from the caching. The requirement R9 "Locality" in Table 1.2 addresses these requirements.

Incorporation of Dynamics (R4)

Traditionally, Quality-of-Service (QoS) mechanisms solve the problem of interference between concurrent transmissions in real-time NoCs by minimizing non-functional dependencies at the cost of less efficient communication protocols. Indeed, the frequently applied, static resource allocation makes communication timing straightforward but results in a significant performance decrease.

However, this is contradictory to the requirements of highly dynamic, contemporary applications in real-time domains that demand allocation flexibility and extensive on-chip reactivity, e.g., advanced driver/pilot assistance systems, industrial robotics, autonomous flight/drive systems. For instance, automated driving introduces a transition of decision making from the driver to the vehicle. Decisions and actions must be taken in the context of a dynamic, continually changing environment, e.g., a situation on the road or weather conditions. Frequently, concurrent execution of complex applications (including high-resolution sensor fusion and data processing combined with machine-learning) is used to address this challenge. At least a part of these functions will be implemented on a many-core system to reduce the number of components and cost. Consequently, NoCs must efficiently host new sets of highly dynamic workloads and the behavior of the system may be influenced by many external factors. Tasks may modify their transmission profiles at run-time depending on the arriving sensor data. Moreover, sets of applications may be initiated dynamically, introducing system modes with changing traffic patterns and mapping or workload profiles, e.g., a convolution of a neural network used for decision making. Similarly, sensor fusion and data crunching bring up a data-dependent execution, resulting in a highly dynamic task behavior and high jitters. Finally, the dynamics can happen due to the transient-errors and mechanisms preventing them such as data re-transmissions or acknowledges. Therefore, the control used for the underlying NoC architecture must protect the system safety against pitfalls resulting from dynamic behavior (e.g., transient overloads, data loss, high transmission latencies and missed deadlines) while delivering the requested performance. The requirement R4 "Dynamics" in Table 1.2 addresses this challenge.

Efficient Verification (R10)

As discussed in Section 1.3,, safety standards enforce separation of the design and specification from implementation. Therefore, designers of the system should provide methods for verification and testing of the NoCs apart from the functioning implementations. In this context, some design decisions may significantly increase the difficulty of the formal verification. For instance, complex mechanisms for assuring high average performance in mixed-critical setups frequently lead to a fine-granular traffic classification which in turn is dominated by the real-time requirements. Consequently, routers and NIs are becoming complex (e.g., extra header information, translation tables, additional logic) making them more difficult to analyze and formally verify. As a consequence, the production costs and necessary efforts increase. On the other hand, it is still essential to share the NoC resources (i.e., allow as many communications as possible) and to assure that transmission latencies are low (i.e., efficient synchronization with minimum impact on senders performance). This additional validation effort should be as little as possible which is addressed in R10 "Verification" in Table 1.2. Note that formal verification must also account for the effects of dynamics described in the previous paragraphs. This incorporation of dynamics may often be complicated and contradictory to other mechanisms in the NoC, for instance, fail-safety. In fact, the load of 50% is already considered to be significant in case of automotive systems exposed to dynamics in system's behavior [118]. Indeed, results from Daimler [118] state: "... it is accepted that the cyclic busload should not exceed 50% (the exact upper bound depends on the OEM)" and later "... In networks dominated by event-driven communication, the theoretical worst case load then overshoots 100% and may reach 500%.". Bosch in [156] supports these claims. Otherwise, an OEM would not be able to provide formal guarantees required by standards.

Safety and Real-Time as Features of the Architecture (R3, R7)

In many architectures proposed by academia, support for real-time and safety features plays a central role, and therefore other practical properties are neglected. It is important to mention that although the automotive electronics market is the fastest growing among embedded systems [98], it is still covering less than 20% of the overall production. Moreover, real-time workloads are usually in the minority of developed and deployed applications. Consequently, from a manufacturer's point of view, the best solution would be to provide a generic platform which can handle simultaneously general purpose, BE applications and, whenever it is necessary, real-time and safety-critical features at low cost. The integration of BE and

real-time (safety-critical) features would allow increasing the production series of SoCs and leave it to the integrator if and to what extent a particular design should incorporate these specific requirements. Consequently, a future NoC architecture should allow safety and/or predictable timing as a feature of the system and not its sole purpose. This challenge is reflected in the requirement R7 "Switch-off QoS" in Table 1.2. Of course, such approach is only feasible if the overhead resulting from the domain-specific mechanisms is low in comparison with the overall production costs of the IP component. Similar considerations apply to methods/mechanisms used for supporting temporal and safety requirements. The workloads in embedded domains may differ significantly depending on applications. For instance, in some setups, the system integrator may only have safety-critical tasks. These safety-critical tasks are usually highly optimized and transmit their data using a regular pattern with a pre-defined transmission size, e.g., feedback loops used in control engineering. However, in other applications such as ADAS, there may be a lot of sporadic and dynamic sensor data which, as discussed before, require different arbitration and policies for efficient work. Therefore, the same generic NoC (in the optimal scenario) should offer designers possibility to select an optimal Quality-of-Service mechanism and strategy for resource reservations depending on the particular deployment. The requirement R3 "Flexibility" in Table 1.2 addresses this challenge.

1.5 Thesis Contribution and Outline

As highlighted in the preceding sections, in many embedded applications high-performance requirements have reached systems with real-time and/or safety-critical elements, e.g., Advanced Driver Assistance Systems or Flight Management Systems. These setups form the major challenge for the real-time NoC architectures as they need to handle the dynamics of system's behavior, to increase utilization/performance, while preserving predictability. The presented work addresses these challenges and introduces a novel, global and dynamic arbitration for NoCs with real-time QoS requirements. The scheme uses key elements of a Software Defined Network (SDN) [86] and adopts them to the needs of real-time NoCs. Consequently, it decouples admission control from arbitration in routers and separates the NoC in a (virtual) QoS control plane and a data plane, see Figure 1.9. The global QoS arbitration is achieved through the adoption of admission control in nodes based on contract-based negotiation between senders. The resource manager must validate if currently available NoC resources are sufficient for the change in QoS before the application receives physical access to the NoC. The first advantage of such an

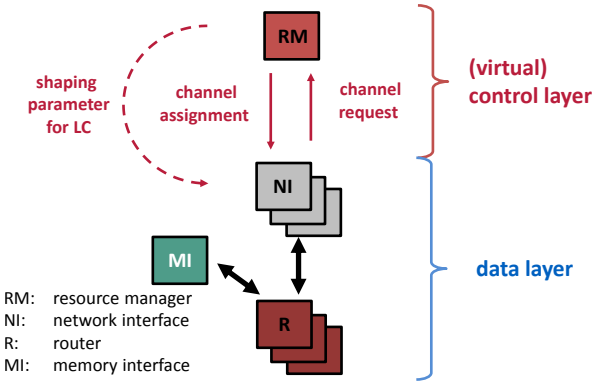


Figure 1.9: NoC extension for decoupling of traffic arbitration in routers and end-to-end, dynamic channel reservations.

approach is that it is aware of the QoS functions of routers. It can adjust to the QoS-functions of routers and no custom QoS-oriented NoC extensions are necessary. Furthermore, the solution allows deployment of arbitration based on the global state of the NoC and defined by currently running senders and the occupied resources. Such scheme proposes a contract-based QoS provisioning at different granularity-levels, e.g. (dynamic) traffic shaping offering a predictable allocation of resources for whole transmissions (constructed of several packets). As a result, QoS arbitration can be optimized in a network-wide fashion depending on the current state of the system. The introduced dynamic adaption to changing system's behavior/mode/environment based on the global system state offers high performance of the NoC arbitration while preserving efficient real-time and/or safety guarantees.

To address the previously described research objectives, this thesis has the following structure:

- Chapter 2 provides an overview of existing mechanisms and solutions for handling safety critical workloads with real-time requirements in NoCs. This chapter also highlights why the existing solutions cannot simultaneously provide high performance and an effective upper-bound on the worst-case latencies

- Chapter 3 introduces the design of the QoS control layer. The description includes the establishment of design requirements, presentation of different

allocation schemes and implementation steps. This chapter also highlights how and why the QoS control layer can solve the problems (i.e., trade-offs between utilization and performance) in most of existing NoCs.

- Chapter 4 focuses on tools and methods required for the implementation of the mechanism. The tools section describes both the timing analysis providing guaranteed worst-case behavior and the simulation environment for the evaluation of performance in a regular operation. Finally, possible implementations are detailed including solutions entirely in software, hardware or as software/hardware co-design.

- Chapter 5 introduces the experimental evaluation including case-studies with commercial and research NoCs. These case studies provide the results from different implementations of the control layer along with the assessment based on benchmarks from automotive and avionic domains.

Chapter 2: A Survey of Mechanisms for Supporting Real-Time in NoCs

The integration of traffic from multiple senders in the same Network-on-Chip results in setups where concurrent transmissions compete for the interconnect resources such as link bandwidth and buffer space. In the real-time systems the most important challenge is adherence to the temporal requirements of applications which dominate other requirements, e.g., high-average performance. Consequently, real-time NoCs must provide mechanisms for supporting predictable arbitration. This means that such networks should offer a predictable upper bound on the worst-case end-to-end latencies. Moreover, whenever the NoC is running as a part of a safety-critical system there is a need to verify that property during the design process and integration, as requested by safety standards. In order to achieve predictable timing, NoC architectures offer two main control features, cf. [37]:

- **flow control** deciding about scheduling and allocation of network's resources, such as channel bandwidth, buffer space, and control state. Flow control is usually realized through control path components, i.e., arbiters, in routers. The duration of the single grant depends on multiple factors. For instance, some requesters may require uninterrupted access to the resource for a number of cycles.
- **admission control** deciding about who and when can initiate the transmission, i.e., when an application can access the NoC. It is usually adjusted in network interfaces of processing nodes, e.g., rate limiters enforcing minimum temporal distance between two consecutive transmissions from the same processing node.

This chapter presents an overview of available methods and mechanisms for assuring temporal properties in a NoC. The presented survey encompasses tempo-

ral and spatial isolation methods. As will be shown, this challenge of providing real-time predictability (and safety whenever relevant) is not trivial as these mechanisms must fulfill multiple design goals at once, as described in the previous chapter. For instance, they must assure predictable behavior and performance simultaneously. Moreover, it is necessary to cope efficiently with the systems' dynamics resulting from application behavior (e.g. activation jitter, transmissions duration) as well as physical environment (e.g. sensor data, off-chip communication). Finally, safety standards for higher safety-levels require verification by formal methods.

2.1 Spatial Isolation of Traffic in Networks-On-Chip

The most straightforward method for achieving temporal predictability in real-time NoCs is spatial isolation, i.e., assigning each of the senders (or sender/traffic classes) its own set of NoC resources, i.e., links and buffers. This is commonly achieved through static load distribution with oblivious routing [37]. When applied, the paths followed by packets during runtime are independent from the current state of the NoC and determined only by the source and destination of the data stream. Consequently, in mixed-critical NoCs, spatial isolation limits the interference between senders of different criticality.

Note however, that in such a NoC all transmissions inherently suffer from the main drawback of oblivious routing - the lack of a load balancing during runtime leading to an under-utilization of resources and performance bottlenecks. In fact, for every oblivious routing algorithm there is a traffic pattern that causes a large load imbalance [37]. Consequently, as it will be discussed in this section, this method although relatively straightforward is contradictory to multiple requirements established in Section 1.2.1. This is especially problematic in mixed-critical and safety-critical setups as discussed in Chapter 1. The spatial separation forbids sharing of NoC's resources (buffers and link bandwidth) between BE and HRTTs. This is enforced by appropriate mapping and path allocation, e.g., [6, 52]. Although the implementation of this method is relatively simple and provides the requested predictability, it simultaneously decreases both hardware utilization and senders' performance in all traffic classes. BE applications are frequently forced to take non-optimal (i.e. longer) routes to avoid interference with HRTTs. NoCs suffer from misbalance in link occupation. For instance, if a HRTT sender does not have any pending transfer its path remains unused even if BE links are heavily loaded, e.g., timing sensitive transmissions from Ethernet port occur rarely when compared to cache traffic of BE tasks [6].

Additionally, strict separation may lead to a segmentation of the NoC and unused/underutilized cores [52]. Moreover, in commercially available NoCs, e.g., Tilera [151] or MPPA [40], nodes are heterogeneous and constructed of processing cores as well as peripherals, e.g., I/O interfaces, memory controllers. As these different hardware units have a fixed position in the system, certain critical communication paths are also fixed and interference between senders with different criticality levels is inevitable (i.e. it cannot be solved by mapping). This endangers temporal predictability of the system and thus its real-time and safety properties. Consequently, the temporal isolation is the only feasible solution in many setups.

Although spatial isolation has significant performance drawbacks it is frequently used in practice due to easy verification. As discussed in [73] or [59], NoC architectures with support for quality-of-service have high production costs.

In [73] an approach based on simplifying the router structure was proposed. As shown experimentaly, by removing the complexity of a baseline router the low-cost router microarchitecture can also approach the ideal latency in on-chip networks. However, as discussed in Section 1.2.1 router simplification may endanger the safety. In [73] the quality of service was implemented through delaying the rate credits that are returned upstream. The evaluation has reported that the proposed architecture can decrease the area by almost 40% and the power consumption by 45% when compared to standard performance oriented solutions.

Such results encouraged research and design efforts for proposing multi-layer networks. These architectures are based on the assumption that under-utilization of the NoC resources can be compensated by simplicity and low costs of router constructions. Consequently, such MPSoCs are supporting several physical NoC layers for spatial isolation.These architectures are popular especially in commercial applications as they simplify the verification and control of the MPSoC.

The Tile64 NoC [151] from Tilera supports five independent networks which are designed to serve different traffic classes: 1) user dynamic network (UDN), 2) I/O dynamic network (IDN), 3) static network (STN), 4) memory dynamic network (MDN), and 5) tile dynamic network (TDN). Four of them (1,2,4,5) are carried out with the same dynamic router architecture supporting dimension-ordered wormhole-routing. The main goals are to preserve the order of messages, offer flow control and reliable delivery for short transmissions from BE and safety critical applications (control-oriented). The STN network is designed as a circuit-switched network for streaming applications. Each streaming sender may set up a route through the network and stream directly without interference.

The MPPA NoC from Kalray [40] offers two layers C-NOC and D-NOC. The first one is designed for worst-case latency sensitive short control messages while the second for bandwidth critical data transmissions. The routers in both layers are

identical with respect to applied 2D torus topology and the wormhole switching. The differences appear at the device interfaces, and the size of router buffers. A similar approach is proposed by [55] where BE and safety critical traffic are separated in two different planes. The layer for HRTTs uses routers supporting the TDM slot allocation done in routers, whereas BE messages are transmitted using a performance optimized router architecture.

Another possible application of spatial isolation is dividing the whole NoC into regions, i.e., assuring through predictable routing and appropriate application mapping that traffic from a specific class does not interfere with other traffic, e.g., BE and HRTTs never share links. For instance, KiloNoC [57] architecture isolates shared interconnect resources into one or more dedicated regions called shared regions (SRs). If QoS is necessary within selected SRs, these SRs may provide an additional hardware support allowing accommodation of senders with real-time and/or safety requirements. The rest of the network is freed from different temporal requirements. These mechanisms are based on prioritizing accesses to virtual channels (buffer queues) in routers. Moreover, there is still the option to switch off QoS in remaining / all regions and by doing so offer maximum performance. A similar approach for the Kalray MPPA architecture was followed by [6]. Authors proposed a methodology for deriving mapping allowing containing traffic from certain applications within selected regions i.e. initiated transmissions can reach only a selected subset of available nodes and therefore traffic of different criticalities never interfere.

2.2 Temporal Isolation of Traffic in Networks-on-Chip

Temporal isolation offers the frequently applied solution to the problem of temporal predictability and safety in the majority of NoCs. According to this approach, senders with different requirements with respect to the interconnect resources are allowed to share links and buffers. Therefore, providing formal timing guarantees for safety-critical traffic in NoCs is usually done through [55]:

- identification of interfering senders,
- temporal synchronization of transmissions competing for shared resources.

In the majority of NoCs, these two actions are considered to be largely orthogonal [37] and therefore separated and designed independently in order to enforce a predictable behavior [100]. The set of interfering senders can be bounded with deterministic or oblivious routing [37] where paths that transmissions may use are selected during the design phase and are static during runtime. Synchronization

of concurrent accesses in time can be done in routers as well as network interfaces with a selected temporal arbitration method. There exist two dominant groups of mechanisms for temporal isolation: Time-Division Multiplexing (TDM) or predictable arbitration in routers. The former offers static guarantees which are independent of the behavior of other synchronized senders. The latter offers guarantees which depend on the behavior of other senders but this interference can be safely bounded, e.g., prioritization of traffic in routers.

However, the market success of a NoC architecture depends mainly on two factors: on an efficient scheduling methodology that does not sacrifice the performance and on a router/switch design that is competitive in terms of area and power in comparison with a conventional NoC architecture. As we will argue in the following of this section, these goals are largely contradictory and each of the methods results in hard trade-offs between performance and temporal predictability.

2.2.1 TDM-Based Networks-on-Chip Architectures

The first group of mechanisms allowing temporal isolation in a NoC is based on the paradigm of Time Division Multiplexing (TDM). In the TDM approach, each sender receives, in a cyclic order, a dedicated time slot for an exclusive access to the NoC [92]. Consequently, TDM-based systems are relatively easy to implement and analyze.

Implementation of TDM in a NoC can be done differently w.r.t link sharing, routing, connection setup, end-to-end flow control and different connection types. An extensive survey of TDM-based NoC architectures is available in [134]. This section concentrates on the significant representatives. The simplest and most straightforward approaches for time-division-multiplexing (TDM) in a NoC utilize circuit switching. In [129] the whole system is considered to be a globally shared resource. Consequently, the NoC architecture provides directed virtual circuits with the same bandwidth for connections between nodes. The establishment of a dedicated communications channel (circuits) through the network is conducted for a predefined amount of time (time slot) and repeated cyclically. A similar approach was proposed by [93]. This NoC architecture offers TDM-based arbitration applied for the usage of virtual channels. Consequently, time slots are assigned based on path and VC pairs. Both works have shown that circuit switched TDM saves resources (e.g. buffers) because once access to the NoC is granted for packets, they can proceed with full speed due to the absence of interference.

However, although TDM-based NoC architectures allow easy implementation and providing timing guarantees, they suffer from severe drawbacks. Firstly, the introduced scheduling is static and non work-conserving i.e. the worst-case laten-

cies depend on the number of synchronized senders and duration of their time slots and not their actual activity during runtime. This causes scalability issues i.e. a high number of synchronized senders (even if they use the NoC sporadically) results in long TDM-cycle and pessimistic guarantees. Moreover, if transmissions are not perfectly synchronized with cyclic resource allocation schemes average latencies are very close to the worst case scenario even when the system is not highly loaded [59]. Therefore, to achieve high average performance, dedicated solutions are necessary, such as an offline generated schedule statically applied to the whole NoC [129]. This is especially visible at the presence of dynamics in the behavior of senders such as activation jitter or variable resource usage (e.g. length of the transmission). Consider the example depicted in Figure 2.1 composed of two applications running in a NoC-based MPSoC, arbitrated using TDM. App1 is composed of three tasks (A, B and C) with precedence constraints and App2 is composed of one task (D). If the behavior of the system would be static and predictable, running tasks could be synchronized using TDM cycles, see Figure 2.1.a), exploiting their periodic activation scheme where the start of an application transmission occurs in a cyclic way whenever it is granted a time slot.

This assumption is non-realistic in most setups. The cyclical repetition of the senders behavior must match the cyclical repetition of the TDM cycles. In the case of sporadic (e.g. coprime) repetition rates of activations and/or bursty memory access patterns of the applications, optimization of the TDM-cycle can only be possible with extremely long TDM-cycles or with extremely short cycles (the divider of 1), as described in the next section. Therefore, even with strict cyclical activation in a fully predictable environment, full resource utilization cannot be guaranteed in practice due to the nature of underlying workloads.

Moreover, whenever tasks expose dynamics in their behavior, even with a small jitter, transmissions are blocked and their execution is delayed for the duration of a whole TDM cycle, see Figure 2.1.b). The activation of task A arrives with a small jitter and misses its slot, therefore tasks B and C cannot start their execution before the next cycle. Consequently, task B is granted access in the second cycle but due to the an activation jitter of Task C, a third TDM cycle is required for App1 to complete its execution.

Resulting architectures are not flexible and the worst-case guarantees can be easily endangered by modifications of running applications. Changes in the toolchain, in the distribution of the load on the cores or by modifications in the software (adding/removing tasks) as well as the hardware may cause additional jitter making the TDM-schedule and / or safety guarantees effectively infeasible. Finally, if an application does not have any pending transfer then its time slot is wasted. This prevents the slack bandwidth from being redistributed to improve the sys-

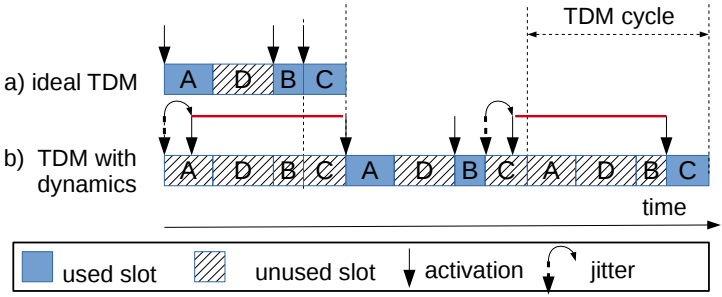


Figure 2.1: Effect of jitter on service guarantees of the applications synchronized with the TDM.

tem's performance. Consequently, TDM-based NoC architectures are moving the main design effort from the NoC arbitration to the software/on-core layer which must optimize the behavior of the whole system with respect to NoC arbitration, thus not visible on the NoC-layer. Using a legacy code without large modifications leads to substantial performance decrease [59] resulting in unfulfilled design requirements.

TDM-based Mechanisms with reduced adverse performance impacts

Such a drastic decrease in the system performance forces designers to optimize setups e.g. by reducing the number of tasks running on the processing nodes, modify the functionality and the number of tasks i.e. the move from requirements resulting from the nature of controlled processes to requirements enforced by the platform. As the overhead of TDM depends on the duration of the cycle, i.e. the number of applications and the size of their time-slots, and not on the frequency of their accesses to the interconnect, the popular solution is to decrease the slot granularity. Using large slots, for instance adjusted for whole DMA transfers from streaming applications, increases the latency of all senders e.g. [129], [129], [56]. Consequently, architectures with shorts slots (e.g. single packet / flit) have been proposed. TDM with short slots shortens the TDM-cycle and decreases the overhead resulting from dynamics. The prominent examples are Aethereal [56], dAElite [134] or Argo [72].

Aethereal [56] architecture supports two physical layers (spatial separation) for accommodation of BE and safety critical traffic. BE traffic can be either transmitted using distributed or source routing with conventional packet-switched router architecture or may use otherwise unused time slots from real-time senders [59]. BE

traffic is also used to set up safety-critical connections, i.e. configure the slot table. The separated physical layer mitigates the overhead for the non TDM-optimized traffic. However, it is done at a substantial hardware cost, i.e., in fact such systems require two separate NoCs with different router architectures. The real-time traffic is routed statically and contention-free using router slot tables. Slots are short and adjusted to single flits. Consequently, it is possible to avoid buffering for time critical traffic class. However, a NoC requires complex calculation of a slot assignment allowing transmissions without interference and dynamics cannot be efficiently captured in time-critical applications. Consequently, recent studies suggest that aethereal architecture is not very cost-effective [55]. dAElite [55] was proposed as a a lightweight version of aethereal. The main goal was to decrease the hardware overhead of the solution by moving the routing from routers to to the network interfaces. Additionally, support for mesochronous networks has been added i.e. NoCs that have a single clock frequency but asynchronous (not in the same phase) clocks. dAElite [133] is another TDM-based NoC with support for short slots which are two words long, and can be further decreased to a single word if necessary. Finally, Argo [72] combines TDM and GALS/Mesochronous arbitration - the network clock of interfaces may have different phases and routers are asynchronous. Similarly to aethereal and dAElite, Argo avoids all buffering and (run time) flow control due to a lack of direct interference in the NoC.

The common drawback of architectures utilizing short TDM-cycles is that short TDM-slots lead to a distribution of longer transmissions over several TDM-cycles, even when the remaining TDM-cycles are not used, which drastically decreases performance. Consequently, transmissions are unnecessarily slowed down even if the NoC is empty. Moreover, too short TDM-slots are undesirable when accessing *state-dependent* shared resources that benefit from spatial locality, such as DDR memories. In case of short TDM-slots, longer transmissions are distributed between multiple TDM cycles and packets from different senders may freely interleave in the memory controller, potentially leading to undesirable timing effects [54]. Therefore, in order to employ short TDM-based setups in safety-critical systems, the sufficient independence between interfering senders must be enforced using *specialized* predictable memory controllers which, to deal with lack of locality, employ a combination of close-page policy and static bank interleaving. This introduces custom hardware increasing costs and power consumption.

Finally, the scalability problems remain i.e. utilization is low as unused slots cannot be handed over and the length of TDM cycles depends directly on the number of synchronized applications. To mitigate these scalability issues, multiplexing of time slots between several channels with independent TDM-schedules was proposed in [59]. The effectiveness of such approaches depends directly on the

number of independent channels as well as their utilization and it requires additional arbitration effort thereby increasing costs of the design and verification. If there are few heavily utilized channels the performance improvement will be low. Moreover, they rely on static budgets which leads to the same problems as in case of TDM-slot's granularity.

Other approaches, such as SurfNoc [150], PhaseNoc [116], employ optimized TDM scheduling to minimize the latency overhead. This is performed by replacing the cycle-by-cycle TDM-schedule with more flexible solutions e.g. domain oriented waves. Although these mechanisms decrease negative side effects of TDM-arbitration, they do not fully eliminate them, providing only an improved solution. Moreover, such TDM-based architectures require complex routers thereby drastically increasing costs of the hardware and power consumption.

2.2.2 Traffic Isolation in the Router

Due to the aforementioned limitations, the static non work-conserving arbitration is not a feasible solution in many setups with dynamic workload. The alternative solution to TDM is based on two main components: local arbitration in routers and rate control for transmissions initiated by processing nodes. This well known solution from off-chip networks [155] has been extended and adjusted for the purpose of the on-chip interconnects e.g. [31, 66, 24]. In such NoCs, transmissions must acquire output ports in routers locally and independently as they progress through the interconnect. The router ports are the only point where queuing occurs and can be modeled as a queuing server. The arbitration between transmissions is performed locally and independently within each of the routers.

Contrary to TDM, these QoS mechanisms do no to prevent interference between senders but rather tend to safely control its effects. Consequently, the offered service for a particular safety-critical transmission depends on the behavior of other streams. For instance, this may not avoid interference due to the other traffic in a router but by selective prioritization it could ensure that a blocked packet on an output port cannot block a link for other arriving packets, i.e., prevent head-of-line blocking.

In the following sections, existing solutions are briefly discussed considering their used priority assignment schemes. The description distinguishes between NoCs in which priorities are assigned offline to the transmissions and therefore static at runtime, see Section 2.2.3, and the real-time NoC in which priorities for a sender may dynamically change during the runtime, see Section 2.2.3. This follows the commonly used classification from related research [30], [92].

2.2.3 Performance Optimized NoCs in the Real-Time Context

The majority of the generic performance optimized NoC architectures apply traffic arbitration done locally and independently in routers. However, their main goal is to deliver possibly low average latencies and high average throughput in the NoC combined with some packaging and cost constraints. These criteria are predominant and drive other design choices e.g. topology, routing, and flow-control.

In such performance optimized NoCs there is no distinction between different traffic classes. All ongoing transmissions compete for output ports (link bandwidth) and virtual channels (buffer space) and receive equal treatment. Therefore, without appropriate quality-of-service (QoS) mechanisms, resources are not reserved in advance, i.e. packets are switched as soon as they arrive. Moreover, some interference cannot be resolved locally by the router's arbiter and requires input from adjacent neighbours. Therefore, verification is difficult as a single operation may involve multiple routers (independent IPs) as well as multiple arbiters within a router e.g. transmission over several routers on a single path with multistage schedulers. This results in a complex spectrum of direct and indirect interferences between data streams which may endanger the system safety [44], [31].

Although recent works, such as [44], [142], from the real-time analysis domain proved that standard NoCs, even with complex two-staged arbitration (e.g. iSLIP) and virtual channels (VCs), are analyzable allowing to compute the worst-case network latency, this analysis is based on several demanding assumptions. Firstly, reasonably tight guarantees can (typically) be provided only assuming the predictable and known beforehand behavior of all potentially conflicting senders. Unfortunately, this is not the case for most of the general-purpose applications.

Secondly, all traffic should belong to the same class (e.g. RT and / or safety-critical) or must be spatially separated from other senders. In order to design such mechanism, one must take into account that, in SoCs with performance optimized NoCs, nodes are usually heterogeneous and constructed out of processing cores as well as peripherals e.g. I/O interfaces, Ethernet or DRAM controllers. For instance, Tilera Tile64 provides 3 Ethernet interfaces and 4 memory controllers while MPPA provides 8 Ethernet interfaces and 2 memory controllers. These different hardware units have a fixed position in the system. Therefore, when applied in real-time setups, certain critical communication paths are also fixed (i.e. defined by the static routing algorithm), e.g., communication from Ethernet controller to DRAM memory. This frequently makes spatial separation unfeasible (i.e. it is impossible to avoid overlap in communication) as already discussed in Section 2.1. Consequently, when applied in real-time systems, performance optimized NoCs suffer from drastic resource overprovisionining or unfulfilled design requirements.

Static Priority Assignments

The work conserving QoS mechanisms in the domain of real-time NoCs frequently utilize static priority preemptive (SPP) scheduling, see Figure 2.2. According to this scheme transmissions from senders with different requirements with respect to interconnect, i.e. different criticalities, are mapped statically to different virtual channels in routers (VCs) to assure functional independence on the data layer. Later, the arbitration is conducted locally and independently in each router. Transmissions mapped to different VCs are preempted. The granularity of the preemption depends on the selected router architecture and switching method. For instance, in commonly deployed NoCs with wormhole switching and virtual channel flow control [37] the preemption can be done on the packet or flit level. The scheduling may optimize different NoC's properties such as latency, buffer sizes and utilization [155].

Exemplary NoC architectures following this principle are QNoC [24], Mango [22] or KiloNoC [57]. The QNoC architecture uses four traffic classes: signaling (for inter-module control signals), real-time (representing delay-constrained bit streams), RD/WR (modeling short data access) and block-transfer (handling large data bursts). Isolation of different traffic classes in routers is done using a static priority assignment scheme. Routers use wormhole flow control, i.e. buffer management and arbitration are performed on flits. Packets are built of a head flit, multiple body flits and a tail flit. Packet routing is resolved upon arrival of the head flit. Resources are released after receiving the tail flit. Packets of different priority are stored in separate buffer queues. Each output port schedules transmission of the flits according to the availability of buffers in the next router, the priority (namely service level) of the pending flits, and the packet based round-robin ordering of input ports awaiting transmission of packets within the same service level. Preemptions, interruptions of currently ongoing transmission of packets with lower priority, are done on the flit level .

An important functional limitation of the QNoC architecture is that it does not distinguish between the different requirements of RT applications within the same traffic class. Consequently, if different senders belong to the same traffic class they must compete with each other and the conflicts are resolved using round-robin scheduling locally in routers.

Another implementation of this general principle is offered by MANGO [22], implementing virtual-circuits. MANGO's ("Message-passing Asynchronous Network-on-chip providing Guaranteed services over Open core protocol (OCP) interfaces") NoC presents an asynchronous architecture where BE and safety critical traffic are separated in each router. Consequently, each router supports two different ar-

biters, i.e. one for BE and one for GS, and RT and BE are mapped to the subsets of available VCs. The arbitration for BE traffic is done through simple packet switched transmissions using source routing. The RT traffic uses virtual circuits - reserving sets of buffers in different routers for end-to-end connections. Real-time traffic (e.g. SRTT or HRTT) is prioritized over BE. Within the same traffic class (BE or RT) a fair (round-robin) or a non-symmetric arbiter can be applied. The latter uses strict priorities to provide non-symmetric latency guarantees.

Similarly to MANGO, KiloNoC [57] applies separation of RT and BE in different virtual channels. Consequently, a baseline router features 25 VCs (therefore 25 priority levels) for accommodating incoming traffic. Each of the VCs in a router has a buffer capable of storing one packet built of four flits. Therefore, each router has 750 buffers belonging to different VCs and 3000 flit slots as it supports 30-ports. To limit this substantial hardware overhead it is possible to switch on and off support for the safety critical applications for selected NoC regions, cf. Section 2.1. If QOS support is off, each router's port serves just one VC per packet class and there are two priority levels (request at low priority and reply at high priority). The required per-port buffering is reduced to 70 flits compared to 100 flits in a QOS enabled router (25 VCs with 4 flits per VC).

As confirmed using formal worst-case analysis methods and simulations, e.g. [31], [66], [24] the prioritization of traffic performed locally in routers (switches) provides HRT guarantees for synchronized applications. However, this method has several drawbacks which limit its applicability. First, the improved work-conserving guarantees offered by this arbitration are possible only under the assumption that packets can be forwarded as they arrive [44], [56] i.e. there is no back pressure and no correlation between routers.To guarantee that, larger buffers must be applied compared to TDM. Otherwise, the propagation of blocking leads to cyclic dependencies and either systems are hardly analyzable or no guarantees can be given. Second, the scalability of these solutions in a mixed-critical context directly depends on the amount of available hardware resources. If the number of VCs is not equal to the number of criticality levels in the system the formal analysis becomes complex and guarantees can be pessimistic or cannot be given at all as it is necessary to aggregate multiple streams within the same class. Therefore, there is a predominant trade-off between real-time predictability and the amount of HW resources used for implementation.

However, the major challenge emerges from the constant increase in the number of applications integrated into a single chip such as the Flight Management System [46] application encompassing multiple tasks with different criticality levels. Note, that the number of VCs is independent of the activities of the senders i.e. number and frequency of transmissions initiated by them. This increases the

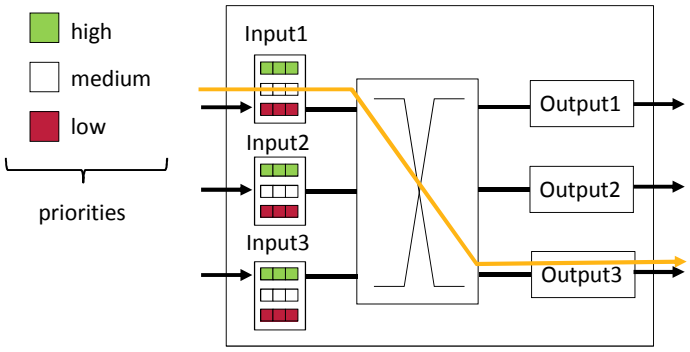


Figure 2.2: A NoC router architecture supporting static priorities which are assigned to independent buffer queues (virtual channels).

cost of design and power consumption because hardware must be optimized with respect to the worst-case behavior of running applications and not their average performance. The worst-case dimensioning is usually overly pessimistic thus drastically decreasing utilization of the system, as in case of TDM. This effect can be observed in case of KiloNoC. Although theoretically it is capable of handling 25 priority levels, it can apply the priority based arbitration only to selected parts of the NoC i.e., other parts of the NoC use simpler routers . This is due to the high hardware overhead resulting from storing streams of different priority in separated queues what increases the resource overhead. Consequently, the overhead is increasing exponentially with the number of supported priority levels.

Finally, priority based arbitration is adjusted to the flow-control units (e.g. packets or flits), not to logical transmissions which are frequently composed of multiple packets. This leads to design challenges known from the TDM-based arbitration e.g. locality, in-order delivery, assumed error rate or burstiness which in many setups may drastically decrease utilization of peripherals and memories. For instance, as discussed in [81], longer DMA transfers could benefit from a locality principle to exploit the stateful nature of the memory e.g. open bank policy in case of DRAM. Indeed, DRAMs have an internal level of caching, which in standard DDR3 modules amounts to 8kB. Consequently, contiguous and aligned 8kB long transfers fully benefit from the caching. In case of SPP-based routers this property can only be assured for the streams with the highest priority. For all others, preemption may happen at any arbitrary moment in time and locality cannot be assured.

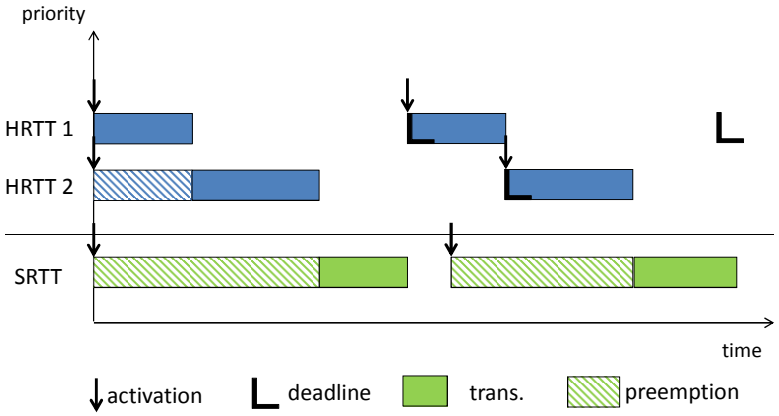


Figure 2.3: Effects of the static priority arbitration performed locally in routers on the latencies of the BE senders in the NoC.

Dynamic Priority Assignments

The priority based solutions are frequently foreseen as solution capable of mitigating the problems of static TDM approaches. However, the SPP-solutions described in the previous section, have a common drawback of treating best effort traffic as a "second-class" citizen. The BE transmissions are preempted as soon as HRRT traffic arrives and therefore suffer from high latency and jitter which is contradictory to their requirements. On the other hand, high criticality traffic is forwarded as soon as it arrives although it has no to little benefit from reduced latency (deadline driven).

For instance, Figure 2.3 presents a mixed critical system where three applications share the NoC. Two of them are safety critical (HRTT1 and HRTT2) and the last one is streaming data (SRTT). Consequently, each sender has a unique static priority (assigned for instance according to a rate-monotonic scheme) and remaining BEs and SRTTs have the same lowest priority. Figure 2.3 illustrates the evolution over time of transmissions in a real- time NoC using a static priority preemptive (SPP) policy, similarly to [24] or [31]. In such systems, SRTTs can progress through the NoC only when HRTTs are not sending data, therefore they are often unnecessarily delayed. HRTTs are scheduled as soon as they arrive, although they usually do not profit from faster execution as long as they are guaranteed to finish before their deadline.

Slack-based resource allocation connected with dynamic priorities is a well known solution from the scheduling theory [92] to this integration challenge, providing high performance for soft-real time applications and BE without violating the timing constraints of hard real-time applications. According to this approach, SRTTs are scheduled whenever the execution of HRTTs can be safely postponed without causing missed deadlines. This is possible whenever a slack is available, which is the time budget between the worst-case response time of a hard-real time application and its deadline. Only a small part of existing research have considered using slack-based resource allocation in the context of NoCs by applying, locally in routers, dynamic prioritization for different virtual channels.

For instance, the "Back Suction" mechanism [43] prioritizes SRTT and BE traffic over HRTT. Consequently, the HRTT is progressing only during idle time when there is no other ongoing transmission as long as the guarantees are not endangered. This is assured by monitoring the buffer space occupied in routers by HRTTs. If buffers are not sufficiently filled, a signal is sent to the upstream router resulting in reverse prioritization of the traffic, so HRTT is prioritized over SRTT and BE. This signal is propagated upstream towards the source. If the buffers for HRRT traffic are filled again than prioritization is once again reverted.

Another example, is WPMC (Wormhole Protocol for Mixed Criticality) [29] which introduces two modes of work for the NoC. Consequently, different traffic classes are separated in different buffer queues and their behavior is controlled before the packets are injected to the NoC. Monitors in NIs observe traffic behavior i.e. message sizes and inter-arrival time. If the system adheres to the predefined behavioral-model, it works in the low-criticality mode in which transmissions of all criticality levels are scheduled based on their typical-case behavior i.e. no differences between different criticalities. When a monitor detects a violation of this behavior, the NoC switches to a high-criticality mode where only streams allocated to VCs of a high criticality are scheduled and all best-effort traffic is dropped by a router to favor the critical packets. This WPMC scheme is extended in [65] for further improving the performance of BEs. The main goal is to allow best-effort traffic to use idling ports of a router even in the critical state instead of dropping it as in [29].

Finally, the mechanism proposed in [141] similarly to [65] and [29] prioritizes best-effort traffic over safety-critical traffic in NoCs as long as its guarantees are not endangered. However, monitoring is done in routers and priority switch is done on demand based on allowed blocking time (slack). Consequently, it is possible to exploit the latency slack of critical senders, while providing sufficient independence among different criticality levels with respect to timing properties. The information about possible slack is contained within the packet header of safety-

critical traffic. Later, the slack value is decremented in each router whenever the interference happens. Packets with low slack values are scheduled before the ones with high slack levels.

The main drawbacks of all aforementioned mechanisms are the high hardware overhead resulting from a custom router design and the high number of virtual channels (i.e. required buffers) corresponding to the number of priority levels in the system. Moreover, these solutions can significantly increase NoC's complexity as they require to propagate the global state of the NoC to the local arbiters in routers. This is incompatible with the principle of local independent arbitration done in separately routers which requires no correlation between local arbiters and thereby increases the complexity of the worst-case timing analysis. Additionally, the configuration complexity increases. Mechanisms such as [65], [29] or [43] require assignments of particular application to one of the existing (supported by NoC architecture) traffic classes. This assignment, when done improperly due lack of well specified traffic behavior or appropriate configuration, can still lead to under utilization of the interconnect as well as a decreased performance of BE and SRTTs sender.

2.3 Comparison of Different Mechanisms

In order to discuss the implications of RT requirements on the NoC design, this section presents an evaluation of the selected NoC architectures frequently used for the deployment of real-time workloads. The examination is done in the context of requirements defined in Section 1.4. Its main goal is to provide an insight into design tradeoffs and open research points in this field. The results are presented in Tables 2.1 and 2.2.

Classification

The discussed designs are divided into two groups: general purpose and real-time architectures. NoCs belonging to the former group are performance optimized (all traffic receives equal treatment) whereas the real-time architectures use mechanisms described earlier in this chapter. Moreover, real-time architectures assume that the load from senders is predictable or can be enforced using **rate limiters (RL)**. RLs define the maximum number of transmissions that a particular sender/processing node can initiate per a given time period i.e. if transmissions arrive too fast they are postponed, cf. Section 1.2.1 for a detailed description. The final granular classification is based on flow-control mechanisms applied in routers

and an applied method for assuring temporal predictability. Consequently, the following NoC architectures are considered:

- CB-CS - Conventional-Baseline Circuit-Switched is a circuit switched architecture which applies round-robin based scheduling between transmissions
- CB-PS - Conventional-Baseline Packet-Switched is a packet switched architecture which applies single stage round-robin based scheduling between transmissions and single shared buffer queue per port
- CB-PS-VC - Conventional-Baseline Packet-Switched with Virtual Channel is a packet switched architecture which applies multi-stage round-robin based scheduling between transmissions with support for multiple virtual channel and thus multiple buffer queues per routers port
- CS-RL Circuit-Switched flow-control and Rate-Limiters in NIs, routers use priority based arbitration including two allocation granularities after which resources must be released: long (logical) transmissions composed of multiple packets, and short transmissions adjusted to the single (or a few) packets
- PS-VC packet switched flow-control and Rate-Limiters in NIs,, support for virtual channels and different router arbiters: round-robin, static priority based and dynamic priority based
- TDM - Time Division Multiplexing realized on different slot granularity (short and long slots), including advanced architecture (advn.) and an optimized slot allocation to cope with dynamic load (e..g. PhaseNoC, SurfNoC)
- Hybrid - frequently referenced architecture with two physical layers: packet switched with round-robin for best-effort traffic and TDM with short slots for RT traffic

Supported Features

The features used for evaluation relate directly to the requirements discussed in Section 1. Note, that "relevant requirement" denotes the primary challenge addressed be the feature, but each feature may simultaneously refer to more than a one problem in the design of real-time NoCs.

- The first feature group encompasses the quality of support for GL and GT traffic understood as efficiency (i.e., pessimism) of the worst-case guarantees. In this case, "high" relates to guarantees which are not far from the actual

performance of the traffic whereas "low" relates to the significant overprovisioning required for providing the RT support.

- The performance of the BE senders in the mixed-critical scenarios relates to the decrease of the average latencies for this traffic which is resulting from application of safety-critical mechanisms.
- Next, features that relate to the dynamics in the sender behavior such as variable activation jitter and variable lengths of transmissions. This allows to assess the additional overhead (delay) which the sender experiences from the applied scheduling mechanisms.
- Later, the influence of these dynamics on other senders in the system is considered. In this context, "yes" relates to the case when a predictable incorporation of dynamics in behavior of one sender may decrease the guarantees of other transmissions through over-provisioning.
- Fairness of arbitration for GL and GT data streams relates to the resource efficient allocation of resources for senders with different real-time requirements e.g. different deadlines, or throughput guarantees.
- Possibility to switch off QoS and hardware penalty for doing so, related to scenarios where the real-time NoC should be deployed in the setup with solely general purpose workloads. (safety as a feature of the design not its main goal)
- Support for scalability, including its hardware and temporal overheads, relates to the quality of safety resource allocation with respect to the worst-case guarantees. The real-time NoC should offer an architecture where the guarantees depend on the actual load of the system and not other static factors e.g. number of integrated senders (temporal penalty). Consequently, the predictable deployment of many senders with different RT requirements should follow at a low cost (hardware penalty).
- Support for locality relates to setups where packets from transmissions must arrive not only timely but also in the correct order to meet the requirements of the peripherals.
- Finally, the two last features relate to the complexity and pessimism of the verification using formal methods, as required by standards for higher safety levels.

NoC Architecture		Real-Time									General Purpose		
Mechanism Type		CS + RL		PS + VC + RL			TDM			Hybrid	CB		
Relevant Requirement	Feature \Arbitration	long trans.	short trans.	static priorities	dynamic priorities	round-robin	long slots	short slots	advn.	TDM+PS	CB-CS	CB-PS	CB-PS-VC
R1	Quality of Support for GL	medium (high avg. latencies depends on trans size)	high (depends on trans. size)	medium (depends on senders prio.; num. of VCs, and number of prios.)	medium (depends on senders prio. and num. of VCs, and number of prios.)	low (depends on rate limiting and num. of senders)	medium (high avg. latencies depends on slot size)	high (depends on slot size)	medium (num. of senders)	high (depends on slot size)	low (depends num.of senders, and msg. size)	low (depends num.of senders)	low (depends num.of senders)
R1	Quality of Support for GT	high	high	medium	medium	medium	high	high	high	high	low	low	low
R6	Performance of BE senders in mixed-critical scenarios	high avg. latencies	medium avg. latencies	high. avg. latencies	low avg. latencies	medium avg. latencies	high avg. latencies	high avg. latencies	high avg. latencies	low avg. latecnies	low avg latecnies	low avg latecnies	low avg latecnies
R2, R3	Sender Penalty for Var. Trans Size	low	low	low	low	low	low	low	low	low	low	low	low
R2, R3	Sender Penalty for Var. Jitter	high	medium	low	low	low	high	medium	low	low	low	low	low
R4, R3	Performance Penalty for others senders in setups with jitter and var. trans	no	no	no	no	no	yes	yes	no	yes	no	no	no
R5, R3	Arbitration Fairness for GL Senders	-	low	low	medium	medium	-	low	medium	low	-	-	-

Table 2.1: Evaluation of different designs for the real-time NoCs w.r.t requirements from Section 1.4, part one.

NoC Architecture		Real-Time									General Purpose		
Mechanism Type		CS + RL		PS + VC + RL			TDM			Hybrid	CB		
Relevant Requirement	Feature \Arbitration	long trans.	short trans.	static priorities	dynamic priorities	round-robin	long slots	short slots	advn.	TDM+PS	CB-CS	CB-PS	CB-PS-VC
R5, R3	Arbitration Fairness for GT Senders	high	high	medium	medium	medium	high	high	high	high	-	-	-
R7	Possibility to switch-off QoS	yes	yes	yes	yes	yes	no	no	no	yes	-	-	-
R7	Hardware overhead in setups with disabled QoS	low	low	medium	high	low	-	-	-	high	none	none	none
R8	Support for Scalability	yes	yes	yes	yes	yes	no	no	yes	yes	yes	yes	yes
R8	Scalability Temporal Penalty	low	low	low	low	low	high	high	medium	medium	low	low	low
R8	Scalability Resources Penalty	low	low	high	high	medium	low	low	high	high	low	low	low
R9	Support for Locality	yes	no	no (yes only for highest prio)	no (yes only for highest prio)	no	yes	no	no	no	no	no	no
R10	Pessimism of formal Verification	medium (low-high)	medium (low)	medium (low)	medium (low)	medium	low	low	low	low	high	high	high
R10	Complexity of Formal Verification	low (high-low)	low (high)	low (high)	high (even higher)	low	low	low	high	low	low	low	low

Table 2.2: Evaluation of different designs for the real-time NoCs w.r.t requirements from Section 1.4, part two.

2.4 Summary

The conducted detailed survey of mechanisms and architectures can be briefly summarized in Table 2.4. The evaluation has shown that there are hard trade-offs in the design of real-time NoCs (with respect to performance, implementation overhead, quality of guarantees) and that none of the existing solutions is fully dominating the spectrum of the desired solutions.

Commonly used performance oriented NoCs use wormhole-switched architectures with multi-stage arbitration for efficiency and scalability. However, they are not designed to meet temporal requirements but rather to deliver high performance on average. In such networks, ongoing transmissions compete for output ports (link bandwidth) and virtual channels (buffer space). Therefore, without appropriate quality-of-service (QoS) mechanisms, resources are not reserved in advance, i.e. packets are switched as soon as they arrive, and all traffic receive equal treatment. Moreover, some interference cannot be resolved locally by the router's arbiter and requires input from the adjacent neighbours e.g. a joint allocation of the crossbar switch and the router output due to a possible lack of buffers at the output (input- buffered router). This results in a complex spectrum of direct and indirect interferences between data streams which may endanger the system safety [122]. Nevertheless, standard NoC architectures still remain appealing for use since they are affordable, fast and flexible [73].

Isolating traffic in space offers each of the senders a dedicated set of resources. Therefore, it offers maximum performance also at the presence of system dynamics (as this sender is the only one using the given path through the NoC). However, although safe and performant, spatial isolation comes at the substantial price of hardware over-provisioning. Hardware resources are not used whenever applications are not transmitting data which is especially problematic in systems with sporadic workloads and high dynamics. These effects increases the production costs as well as power consumption of the MPSoC.

In order to mitigate the shortcoming of spatial isolation, temporal isolation has been proposed. Two widely established solutions are Time-Division Multiplexing (TDM) (e.g. [56]) and source rate-limiting (e.g. [155]). TDM based architectures consider the network as a single shared resource protected by a global (TDM) arbiter. Each application or transmission is granted, in a cyclic order, a dedicated time slot to have exclusive access to the NoC. The size and number of these slots (i.e. the TDM schedule) is defined during the system's design phase and optimized for the worst-case behavior. Although TDM offers a relatively simple analysis and implementation, it drastically reduces the system utilization whenever the applications do not behave according to the predefined static scheme used for the schedule.

	Performance NoC	Spatial Isolation NoC	Static Temporal Isolation NoC (e.g. TDM)	Dynamic Temporal Isolation NoC (e.g. SPP)
Performance	✓	✓	✗	✓
Implementation Overhead	✓	✗	✓	✗
Work-conserving	✓	✓	✗	✓
Safety/ Predictability	✗	✓	✓	✓

Figure 2.4: Comparison of different NoC architectures with the special focus on safety and performance in the presence of dynamics.

Dealing with transient errors, arrival jitters, changes in communication volume, or conditional executions of synchronized senders, results in average latencies close to the worst-case or overly complex schedules which are hard to calculate and maintain. These effects happen even in lightly loaded systems introducing a major overhead squandering the performance advantage of MPSoCs. Furthermore, the overhead of TDM depends on some static factors i.e., the number of applications and the size of their time-slots, thus it rises the scalability problems known from the bus-based interconnects. Therefore, although TDM supports independent verification of each communication's runtime behavior, it comes at the significant performance penalty whenever the system exposes dynamics in its behavior.

In contrast to TDM, rate control avoids the drawbacks of a fixed time schedule. This scheme is based on two main components: local arbitration in routers and rate control for transmissions initiated by senders. Hereby, the local arbitration in each router assigns resources independently (link-bandwidth and buffers) to packets traversing the NoC. Rate control is applied in the network interface (NI) of a processing node and limits the maximum number of packets sent per sender. Consequently, it provides an upper bound on the interference a transmission can experience in the network. Additionally, transmissions can be isolated in dedicated buffer queues in routers (virtual channels - VCs). VCs enable arbiters in routers to further separate senders with different QoS requirements by a priority-based arbitration. However, the multi-stage arbitration and resulting transient run-time effects require a complex corner-case analysis, i.e., obtained results are either overly

pessimistic or difficult to verify. Additionally, as all the arbiters must be optimized and adjusted to the formally verified worst-case behavior, the performance overhead in highly-dynamic setups is significant. For instance, as a local arbiter cannot control the number of simultaneously transmitting senders, buffers in routers' ports must be large enough to accommodate all incoming packets. Otherwise, the packets can be lost (overwritten) or cyclic dependencies between schedulers may endanger the safety (e.g. propagation of blocking). Similarly, the settings of the rate regulators are static during runtime which contradicts the required dynamics of the system, i.e., it is not possible to change resource allocations along with the varying system state without loosing safety guarantees. Note that performance penalty increases along with the complexity and inherent dynamics - as in the case of the TDM-based arbitration.

Consequently, the static and dynamic resource allocation schemes, as used in current safety critical systems, are no longer sufficient to provide the needed level of performance without endangering the safety guarantees.

Chapter 3: The QoS Control Layer

This chapter introduces a new method for operating a NoC which permits to simultaneously satisfy performance and real-time/safety requirements of modern MPSoCs in the presence of highly dynamic workloads. To achieve this goal settings of the QoS mechanisms are adjusted at runtime depending on the global state of the system. The base functionality is delivered through a novel, *global* arbitration layer using key elements of Software Defined Networks (SDN) [86] and adopting them to the requirements of real-time NoCs [78, 81]. The proposed scheme is based on decoupling of admission control from traffic arbitration and separates the NoC in a (virtual) QoS control layer and a data layer, see Fig 3.1. The control layer is used to modify settings of admission control mechanisms (i.e., who, when and for how long may use the NoC) depending on the state of the system (i.e., who is active and which resources are currently occupied). These actions are carried out using a protocol-based negotiation between senders, i.e., providing a validation method to verify if the currently available NoC resources are sufficient before the application is granted physical access to the NoC. The synchronization can follow through a central scheduling unit, e.g. [78], [81] or directly through broadcast or multicast protocols, e.g., [77].

The introduced decoupling of the admission and flow control is appealing in several aspects. The major advantage of the proposed approach comes from the fact that the routers can be agnostic to the global QoS functions. Therefore, there is no need for custom QoS-oriented NoC extensions which can be costly in terms of area and power, e.g., [56], [116]. Additionally, the proposed solution allows deployment of a sophisticated contract-based QoS provisioning (e.g., dynamic priority-based arbitration) without introducing complicated and hard to maintain schemes, known from hardware arbiters in real-time routers, e.g., [43]. Consequently, it allows the application of commercially available performance oriented NoCs in real-time domains. Otherwise, these architectures introduce complex interdependencies between senders leading to pessimistic worst-case guarantees or lack of formally assured safety [142], [44]. Furthermore, using the knowledge about the current (global)

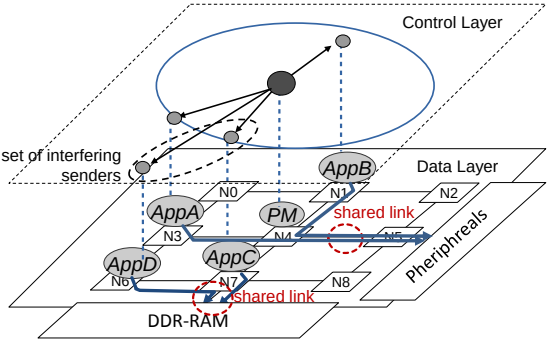


Figure 3.1: Illustration of a QoS control plane and its operations in a NoC domain.

state of the system permits to efficiently incorporate dynamics in senders' behavior while providing real-time and/or safety guarantees. Consequently, the proposed control layer offers performance known from standard NoCs and QoS necessary for real-time applications without the need for specialized infrastructure.

The rest of the chapter is structured as follows: Section 3.2 gives an overview of the related work regarding Software Defined Networks and their applications to NoCs as well as real-time and/or safety-critical systems. Section 3.3 introduces main concepts of the control layer, followed by an overview of the centralized architecture in Section 3.4. Figure 3.2 presents an overview of possible applications of the control layer, which will be discussed in the remaining sections of this chapter. The main purpose of the mechanism is allocation of resources in the NoC. For achieving this goal, the control layer may use different resource allocation policies which are discussed in Section 3.6. Additionally, the clients and RM may be used to build the interface between NoC and the on-core schedulers what increases efficiency in utilization of nodes, see Section 3.7. Moreover, as discussed in Section 3.8, RM may optimize scheduling of NoC accesses with respect to the requirements of other connected peripherals and memories. Finally, Section 3.9 provides a summary which concludes the chapter.

3.1 Baseline NoC Architecture

The proposed solution is built on top of an existing NoC and is largely orthogonal to the underlying interconnect. As there exist a wide selection of possible

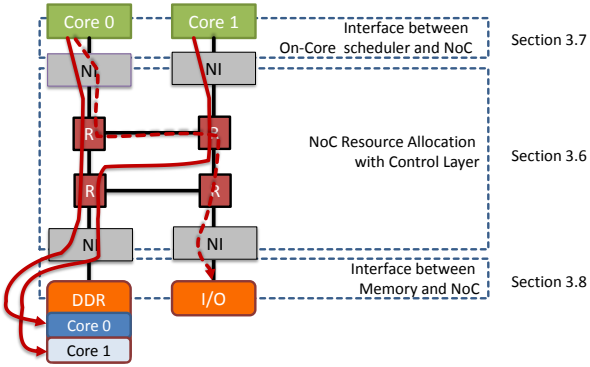


Figure 3.2: Applications of the control layer in the NoC-based MPSoC.

solutions (see Chapter 2), in this section we present the underlying NoC (router and NI) on which we focus throughout the remainder of this document. The description will highlight all essential features and requirements which must be considered for porting the mechanism to other NoC architectures. The primary goal is to build a solid theoretical basis using the in-depth discussion with the selected NoC example and, without losing the generality, to allow a fairly straightforward application of the solution to other architectures.

3.1.1 Router

The baseline router design is largely based on the generic architecture proposed by Dally [37], which is frequently cited and applied in the related work. Figure 3.3 presents its block diagram. The datapath of the router is responsible for storing and forwarding the data and is built out of the following modules: *input ports* with *VC buffers*, a *switch* and a set of *output ports*. The remaining modules belong to the control functionality and are responsible for allocating available resources (i.e., buffers and links/output ports) to the concurrently running transmissions.

Router arbitration supports prioritized virtual-channels (VCs) (cf. [24]) and wormhole switching (cf. [37]) where packets are decomposed into flits. These flits constitute the granularity of transmission and arbitration. Each packet is constructed of a (one) header flit (including address information), a variable number of body flits, and a tail flit. An arriving header flit is stored into the appropriate FIFO queue (depending on the priority resulting from statically assigned VC) and triggers the

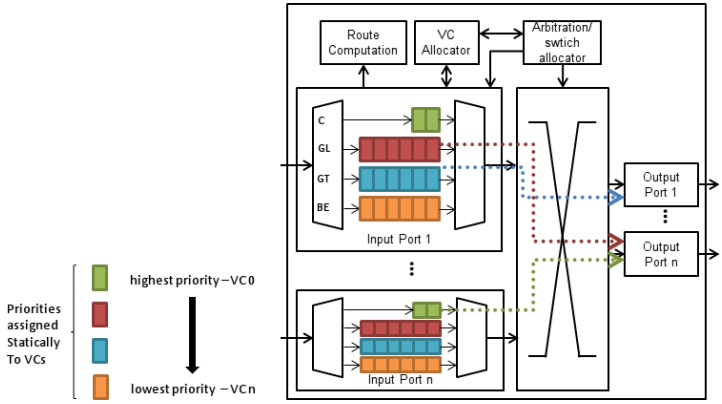


Figure 3.3: The exemplary baseline architecture for the NoC .

arbitration. Forwarding of the tail flit releases the resources.

Each virtual channel has an assigned fixed static priority. Buffering happens in input ports, and buffers for different virtual channels (priorities) may differ in length, i.e., asymmetric buffers. The routers schedule transmissions according to:

- the availability of buffers (for a given priority) in the next router (flow control/backpressure signals),
- the priority of the particular VC, and
- the packet-based round-robin between the input ports for packets using the same VC.

The transmission of a packet is preempted at the flit level (i.e. flits cannot be preempted) as soon as a new packet with higher priority arrives at the router. Later, a blocked transmission can resume after all higher priority packets are served. A deadlock-free, deterministic, source routing policy is assumed.

3.1.2 Network Interface

To assure predictable access patterns of integrated senders, network interfaces use rate control [147], [155], [41]. The mechanisms for rate control are a commonly applied safety measure in many real-time setups as they safely bound the number of initiated transmissions in NoC (accesses) as well as their length. According to

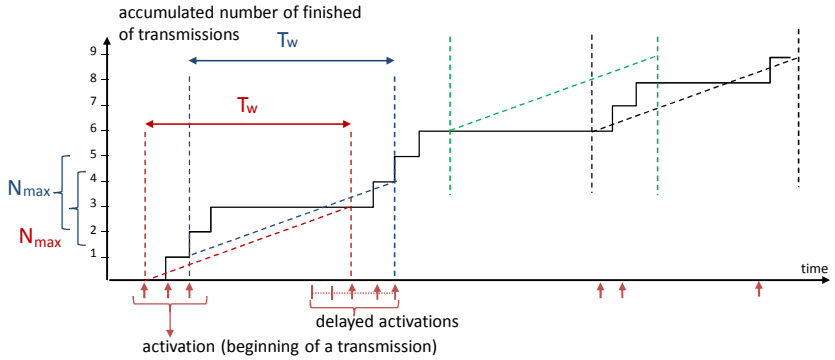


Figure 3.4: Rate control for providing safety guarantees in real-time NoCs.

this method, at each source node, a monitor - **rate limiter (RL)** - regulates the pattern of injected traffic. This is realized through adjustments of two parameters: window length (T_w) and bandwidth quota (N_{max}).

At each cycle, the rate limiter compares the length of a pending packet to the available bandwidth quota plus the amount of data (e.g., packets or flits) sent during the previous period equal to the window length (in cycles or ms). If not higher, the packet can be injected to the NoC; otherwise, it must wait until the budget will replenish. Figure 3.4 presents an example illustrating the principle of such flow regulation at the source. The X axis depicts the trace of accesses from a node to the NoC whereas the Y axis depicts the accumulated number of finished transmissions. In the figure, within each time window (T_w) there may be only three ($N_{max} = 3$) conducted transmissions. When the second burst of three activations arrives too early, it is postponed and may be executed only after the first window finishes. Related research, e.g., [108], has also reported that the fine-granular representation of activation patterns is generally possible with the means of minimum distance functions. The fine-granular rate control offers a better latency compared to the classical periodic (coarse-grained) monitors and better control of the overhead in terms of necessary processing time/power and memory.

Consequently, extending Network Interface with monitors using the rate limiters is a well-known solution for supporting quality-of-service in NoCs e.g. [148], [135], [23], [44]. This method owes its popularity to a relatively easy implementation (no need for custom router architecture) as well as flexibility (may cover different activation patterns, no need of "predictable execution models"). Therefore, it is mainly suitable for BE traffic from processors with caches and as such already

available in many NoCs on the market, e.g., MPPA from Kalray [40].

However, this solution has also a drawback. It is trading performance for temporal guarantees, i.e., comes with a performance penalty. The rates are adjusted statically according to the worst-case scenario during the design or startup time, i.e., designer must assume that all senders are running simultaneously with maximum possible transmission arrival rates and adjust limiters appropriately. Therefore, this approach is not work conserving and sacrifices the interconnect utilization whenever senders expose dynamics in the execution time, release jitter or communication volume, i.e., whenever the system is not highly loaded.

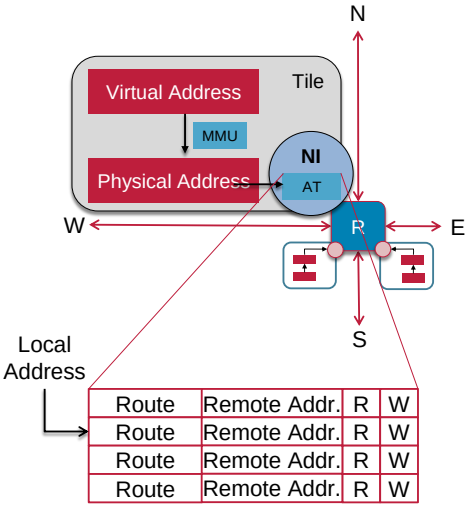


Figure 3.5: Address Translation in NI for providing safety guarantees in real-time NoCs.

Additionally, in more complex/larger setups (e.g., many senders per node, mixed-criticality) a NI may support address translation and work similarly to the functionality of a Memory Management Unit (MMU) or a Memory Protection Unit (MPU)[1], see Figure 3.5. Consequently, the mapping of the local physical address within a tile (processing node) is performed with the address translation tables. Each record in these tables may include the route (which a transmission should take), access rights at the target (e.g. read or write permissions), and the physical address at the remote node.

[1]Note, that an application of such unit in the NI is orthogonal to the work of a MMU/MPU units running in the tile.

Such mechanism offers several significant benefits, in the context of safety. Firstly, it isolates the selected parts of a NoC, what is especially important for the spatial isolation of traffic and mixed-critical setups. Secondly, it separates address spaces for different tiles in order to provide backward compatibility and higher flexibility of integrated software. Finally, overlapping address spaces can be used for the accommodation of the core-to-core communication.

3.1.3 Traffic Characteristic

This section discusses the typical NoC traffic patterns resulting from design constraints in the real-time and safety-critical domains. To achieve temporal predictability and shorter worst-case execution/response times (which can be proven with formal methods), designers distinguish between two traffic groups: short and long transmissions. The former group consists of memory accesses, typically cache-lines, from applications compiled with regular toolchains. They originate from both, real-time and best-effort, traffic classes with a typical metric of worst-case and average latency of a single packet. The latter group contains messages composed of several packets which are usually sent by real-time and streaming applications. Such senders are usually optimized with respect to controlled processes and hardware (*control-oriented* real-time applications). The typical performance criterion for such applications is the worst-case or average throughput. In the following part of this section, we describe both groups in detail.

Long Transfers and Timing-Critical Traffic

The main goal of systems with hard real-time requirements is to decouple on-core interference from scheduling of accesses to shared resources, e.g., shared main DDR-SDRAM memory. Such decoupling simplifies the verification process which is otherwise costly and complicated. Designers of applications in real-time domains, e.g., automotive and avionic, frequently achieve this goal through predictable execution models (e.g. [113]) applied to tiles with local memories (scratch-pads). Figure 3.6 presents a "classic" example of this approach - "read at the beginning and write at the end" which is applied in both avionic and automotive domains [112, 114]. According to this scheme, the on-core execution of an application can be divided into three phases: memory read block (data *Acquisition*), actual execution (*Execution*) and memory write block (*Restitution*). During the acquisition phase, a task or an application accesses the external memory for receiving data and/or necessary code. Later, during the execution phase, the application/task can proceed sporadically accessing the shared resource to ensure data consistency. Fi-

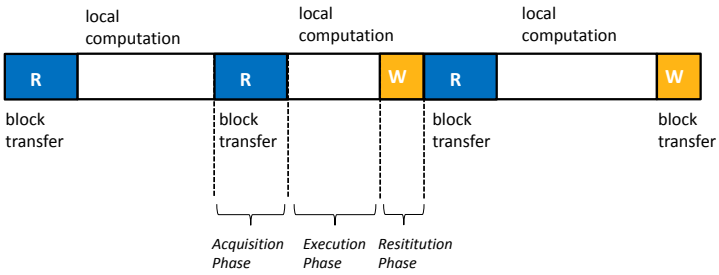


Figure 3.6: Model enforcing predictable on-core execution in safety critical domains [49].

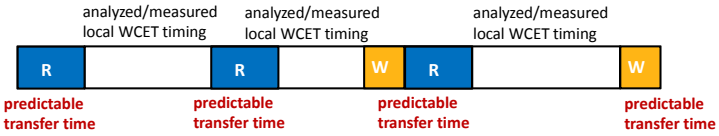


Figure 3.7: Predictable timing through idependent reservation of NoC and on-core resources [49].

nally, it may write the results at the end for further usage or integration by other system components. The introduced separation of the execution from the communication is performed earlier during the design phase i.e. white-box approach [113], [126]. However, in some setups/scenarios, it may not be possible or efficient to load all required data during the acquisition (read) phase due to the nature of the performed operation, e.g., database search. Therefore, tasks must still be capable of conducting "sporadic"accesses to the main memory (e.g., loading specific records from the database in case of drive management). As a result, the underlying NoC's infrastructure must frequently support a combination of long and short accesses.

The main advantage of the predictable execution model is decoupling of local (i.e. within a tile) and global (e.g. NoC, DDR-SDRAM) resource reservations, see Figure 3.7. The worst-case timing can be verified by measuring two factors: local (on-core) worst-case execution time and accesses to shared resources. As the latter factor includes latencies of NoC transmissions, the designer must determine through measurements and analysis that there exist an upper bound on the transfer latency. The efficiency of this evaluation can be supported by design, e.g., on-core or peripherals schedulers and architecture. The next sections discuss these choices for NoCs in detail. The second group of applications requiring throughput

guarantees (soft- and hard-) includes signal-processing and streaming senders. As discussed in [37], their requirements are usually directly related to user perception, e.g., video and audio codecs, or resulting from standards e.g. DVB, DAB, TSN, AVB.

Consequently, timing-critical traffic in such setups often exhibits regular and predictable access patterns, e.g., periodic communications with jitter. Moreover, initiated transmissions are built of multiple packets what promotes the usage of DMA engines, e.g., video traffic or memory read accesses during the acquisition phase. The effectiveness of DMA transfers may be limited by the size of the local memories (in tiles) as well as communication costs resulting from interference with other senders using the NoC. In many setups, the regularities in the behavior of real-time senders are exploited to improve utilization of resources. Moreover, some peripherals, such as DDR-SDRAM memories, are state dependent, i.e., their response time depends on the history of previous accesses. Therefore, clustering of transmissions combined with a known memory controller behavior assures a known access sequence, for further decrease of response latency.

Short Transfers and Best-Effort Traffic

Short transmissions (e.g., single packet or flits) are usually originating from control-oriented real-time applications as well as best effort senders running on processors with caches. Such transfers are sensitive to latency, i.e., the reaction time is critical for the worst-case or average performance [37, 43]. In many architectures, cache misses increase the worst-case execution time of a task and thus a load of a processor. Since it is only possible to limit the number of misses conservatively (see overview paper [152]) the miss-times must be included in the WCETs several times. Therefore, they conservatively increase the worst-case load. Furthermore, the characteristic of traffic from general-purpose, best-effort applications is usually unknown, which may endanger the deadlines (i.e., temporal properties) and/or safety. If the aggregated traffic exceeds the available bandwidth the real-time requirements of other senders using NoC resources may be endangered. The popular and frequently applied method to solve these problems is traffic shaping using rate limiters. This scheme is discussed in detail in Section 3.1.2.

3.2 Software Defined Networking

The main concept of the proposed resource manager and control layer is derived from the global management scheme for off-chip interconnects - Software-defined Networking (SDN). This section briefly revises and summarizes related research based on [137] and [86]. Key concepts of SDNs rely on the division and

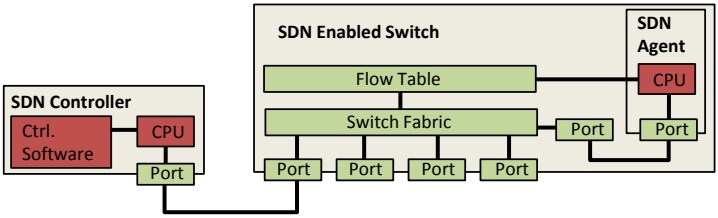


Figure 3.8: Main SDN components in a switch, based on [137].

separation of interconnect's mechanisms into control and data planes. The data plane is responsible for the forwarding of transmitted data, e.g., packets or frames. The control plane controls, through protocol-based synchronization, the behavior of routers and adjusts it depending on the global state of the system.

3.2.1 SDN Principles and Real-Time Systems

The exemplary SDN scheme, taken from the Ethernet domain, is presented in Figure 3.8. Components belonging to the data plane are marked with green whereas the control plane is marked with red.

Traditionally in a SDN, the control layer is build of a centralized SDN controller and agents in the network switches (red components in Figure 3.8). The data plane encompasses remaining functionality of the switch (green components in the figure). The role of the SDN agent within a switch is to monitor the communication passing the switch, synchronize it with the SDN controller and, if necessary, update/modify the rules used for traffic arbitration. To do that, the SDN agent uses a flow table inside the switch which stores settings for forwarding operations performed by the switch's data plane. For instance, each incoming traffic stream (frame or packet) can be identified by source and destination MAC addresses and/or port numbers. Consequently, the flow table may contain settings deciding about the output port to which the particular packet (transmission) should be forwarded.

The SDN controller is usually realized in the form of a software library (also referred to as network operating system) running on a dedicated core. The communication between SDN agents and the controller follows in the same network in which general communication is done. *The SDN interface* defines the communication protocol used for synchronization between the SDN controller and the switches/routers.

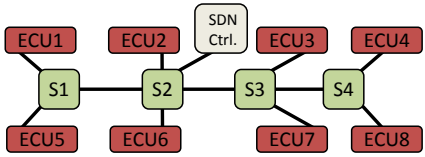


Figure 3.9: An exemplary topology for automotive Ethernet enhanced with a SDN controller, based on [137].

A plethora of mechanisms supporting SDN exists for off-chip networks with a predominant focus on Ethernet. A comprehensive overview of SDNs (including standardization, interfaces, controllers, network programming languages, and applications) is provided in [86]. The primary research focus is placed on optimization with respect to average system performance. Consequently, the schemes are either hardly analyzable or no guarantees can be given. For instance, the most prominent SDN interface OpenFlow [102] relies on TCP-based communication. Therefore, it suffers from complex handshake and flow-control schemes which significantly increase the worst-case latency of real-time traffic [137] which in case of Ethernet is typically UDP-based. Nevertheless, SDN Ethernet protocols have been analyzed in [12] and [137].

The challenges in porting of the SDN principles to the real-time and/or safety-critical domains, such as automotive and avionics, will be discussed using an exemplary topology from the automotive domain which is presented in Figure 3.9. In this context, the role of the SDN controller would be to manage and control arbitration settings (e.g., routing or rate control) in switches depending on information about the global state of the entire network, i.e., who is sending and which resources are occupied. The re-configuration includes admission control and run-time reconfiguration. The former limits the link load and therefore network congestion by bounding a number of NoC accesses (blocking or rate control) for selected senders. The latter can be used for recovery from faults, e.g., through bypassing a failed switch or link.

A single centralized point of synchronization through the SDN controller could be a performance bottleneck. However, it has also significant advantages, especially in the safety-critical domains. Firstly, it simplifies the synchronization protocol, e.g., (rapid) spanning tree protocol or shortest path bridging. In contrast,decentralized systems frequently require lengthy and complicated protocols for assuring system coherence. It is so because the communication's participants must firstly agree on a state of the network before they can take actions. In case of a single SDN

controller, all actions are serialized in this node what simplifies the process of worst-case verification.

The application of SDNs in safety-critical domains, e.g. automotive or avionic, requires providing strict timing requirements and verification methodology for the system, cf. Chapter 1.4. Consequently, the time necessary for admission control and re-configuration including actions of the SDN controller and agents must be bounded in time (worst-case behavior). Verification in such real-time systems is frequently done with formal worst-case analysis methods and/ or simulation-based approaches. The disadvantage of the latter is the lack of guarantee that the simulation exposed included all corner cases leading to worst-case behavior. On the other hand, the former gives safe upper bounds on a system's worst-case behavior which can be however slightly pessimistic (includes scenarios which won't happen in practice).

In [12] a formal analysis of the communication in a SDN utilizing OpenFlow protocol has been proposed using the network calculus. These work models the behavior of a SDN switch regarding delay and queue length boundaries. Next, it provides the analysis of the buffer length of the SDN controller and the SDN switch by modeling them as single resources (network calculus servers). The work has proved that indeed it is possible to provide a formal upper bound on the transmission latencies in a SDN network.

In [137] authors proposed an alternative SDN analysis focusing on automotive Ethernet using Compositional Performance Analysis. Switches have been modeled considering their internal structure (e.g., ports, multiple traffic queues). Moreover, for both, SDN controller and SDN agents, a limited amount of processing resources (CPUs) has been assumed. Moreover, in contrast to [12], the latency of network re-configuration also has been considered for the worst-case timing derivation. This refers to the temporal delay required by the SDN controller to reconfigure the network by distributing new forwarding rules to individual switches and wait for their acknowledgment. Finally, the overhead of introducing SDN has been evaluated, i.e., the impact of SDN traffic on non-SDN traffic.

Finally, some related research, e.g. [96], [154] considered using queueing theory for performance evaluation. These methods, although useful for identifying bottlenecks, offer only probabilistic performance guarantees given by queuing theory. Therefore, they are not capable of deriving worst-case metrics required in real-time safety-critical domains. At the moment of writing, there is no related research considering application of SDN in NoCs considering real-time and/or safety objectives.

3.2.2 SDN mechanisms in NoCs

Existing NoC architectures, implementing the concept of SDN, concentrate on the reconfiguration of independent NoC switches to optimize the average performance by adjusting the scheduling arbitration during the runtime. Consequently, they offer no guarantees for the worst-case behavior. Next, the most significant contributions are described in detail.

In [35] authors present a software-defined on-chip network (SDNoC) in which arbitration logic in routers is software controlled, and SDN re-configures the NoC for improving the average latencies. SDN agents are fully embedded within routers and used to exchange control messages. Consequently, routers are significantly more complex than the baseline implementation. The control capabilities include routing and link load. The former can be used for application of fail-over mechanisms. These allow switching to a redundant or standby hardware component upon the failure. SDNoC has no central management unit. Resource allocation is done directly by sending applications which are adjusting headers of packets issued by them. This limits significantly the applicability of these mechanisms in hard real-time domains, i.e., all senders must be certified to the highest adequate level. Additionally, no solutions for conflict resolving between requests from different senders is proposed.

In [124] authors present an alternative SDNoC architecture in which they are also directly applying principles known from off-chip SDNs to the on-chip interconnect. Consequently, switches are enhanced with agents and configuration tables which decide about routing whereas the SDN controller is running on a selected network node. This leads to significant hardware overhead (similar to [35]) as router complexity increases with the number of senders and synchronization scenarios. Finally, in [18] SDNoC is realized in the form of a hybrid hardware/software approach. Firstly, authors introduce spatial (physical) isolation between the control network and the data network. Next, SDNoC is controlled by a centralized network manager (NM) which is running on a selected NoC node in the form of a software library. Finally, SDNoC reduce buffering in the data layer.

There are significant differences between the solution proposed in this work and the approaches which are described above. The majority of the related work applies a straightforward approach for porting the off-chip SDN ideas and mechanisms to the on-chip interconnect. Consequently, the SDN-controller adjusts the settings of routers as it is done in the off-chip networks. This limits efforts required for porting the SDN idea but disregards the specifics of the on-chip interconnects. As discussed in [73] and [37] the constraints imposed on an on-chip interconnect differ significantly from off-chip networks, e.g., small size of routers, short laten-

cies of memory traffic. As a result, the direct adoption of the principles from a conventional SDN off-chip network switch microarchitecture results in an overly complicated design. Such NoCs increase not only production costs (concerning area and power) but even more importantly latencies of traffic (typically memory accesses).

In contrast to the related architectures ([35], [124], [18]) the control layer proposed in this work controls admission of the traffic at the source, i.e., checks the availability of the interconnect resources before the sender can obtain physical access to the interconnect. Therefore, no router modifications are necessary, and agents can be implemented as a straightforward extension of the rate control mechanism available in many architectures. Moreover, the described solution is focused on assuring temporal guarantees for synchronized real-time and/or safety- critical senders. Currently, in the context of NoCs, client-server admission control schemes are only applied to improve average latencies of synchronized transmissions by limiting the access rates to the frequently requested peripherals such as DDR-SDRAM memories, e.g., [147], [76].

3.3 The Main Concepts of the Control Layer

In real-time systems using NoCs, synchronization between interfering transmissions can help to run the network with significantly fewer resources and still be able to guarantee temporal properties of the system and, whenever required, its safety. Achieving these goals includes handling the effects of both backpressure and head-of-line blocking in routers, where switch arbitration between packets/flits is performed [76], [147]. To provide service guarantees, e.g., an upper bound on the worst-case latencies or guaranteed throughput, the architecture must allow bounding direct and indirect interference between *interfering transmissions* - transmissions which overlap in at least one router on their path from source to destination and therefore are sharing NoC resources, i.e., link bandwidth and/or buffers in routers. For doing so, applications are grouped in *synchronization scenarios*, i.e., sets of senders which may mutually influence their execution times, e.g., through concurrent accesses to shared interconnect resources. Assuring temporal guarantees for synchronization scenarios, i.e., offering a *predictable NoC*, requires reserving enough resources so that traffic requirements from all senders are met, cf. Chapter. 1.

To achieve the goals mentioned before, it is required to start with decoupling of the admission control / QoS configuration from the system execution, using key principles of Software Defined Networks. This is reflected by the architectural

measures introduced by the proposed *control layer*. The existing NoC is treated as a *data layer* (communication domain) responsible for switching of particular packets. The *control layer* (model domain) responsible for a safe *admission control* is built on top of it. Admission control is understood as a validation process performed at runtime before communication is established to see if the currently available NoC resources are sufficient for the particular transmission. Consequently, the main functionality of the control layer encompasses model-based analysis methods which are used to establish the settings for monitoring and isolation mechanisms. Both control and data layers are coupled with a protocol based synchronization, i.e., contracts, allowing *resource reservations* for senders.

In case of a switch-based interconnect, these reservations must include all routers on the end-to-end path through the NoC (i.e., more than one network resource). Consequently, for each transmission, it is possible to define a *virtual resource* - set of real resources belonging to different network components which must be selected and allocated to satisfy its requirements. This is accomplished through *connections* managed by the introduced control layer, i.e., contract-based end-to-end resource reservations for a given sender-receiver pair on the selected path with a selected QoS provisioning. The *QoS provisioning* defines the minimal set of resources requested for connections (reservations) including at least the type and desired upper time response of the NoC depending on the particular synchronization scenario. For interfering transmissions, the *response time* translates to the latency for a particular volume of data and rate settings for rate limiters (traffic shapers).

Connections (for each synchronized sender) must be synchronized and arbitrated at runtime. The arbitration is based on the global state of the system, i.e., *coordinated reservations*. The *global state of the system* is defined by the number of currently running applications and their requirements for the shared NoC resources. Note, that both factors mentioned earlier can change during runtime depending on the dynamics of the system itself as well as the physical environment in which the system is used, e.g., a situation on the road, cf. Chapter 1.

Conceptually the proposed resource arbitration scheme introduces the model domain architecture, see Fig. 3.10. The deployment of the control layer is realized using multiple *analysis engines* responsible for the evaluation of different aspects of the resource allocation and admission control. The analysis can be done to optimize various aspects of the system's work (e.g., safety, security or performance). Moreover, the analysis engines can assign resources according to the pre-defined static (e.g., Time-Division Multiplexing) as well as dynamic allocation schemes and work-conserving schedulers (e.g., round-robin, priority based policies or even dynamic priority assignments). Summarizing, the behavioral model defined by connections for a particular sender on a specific node is negotiated with the control

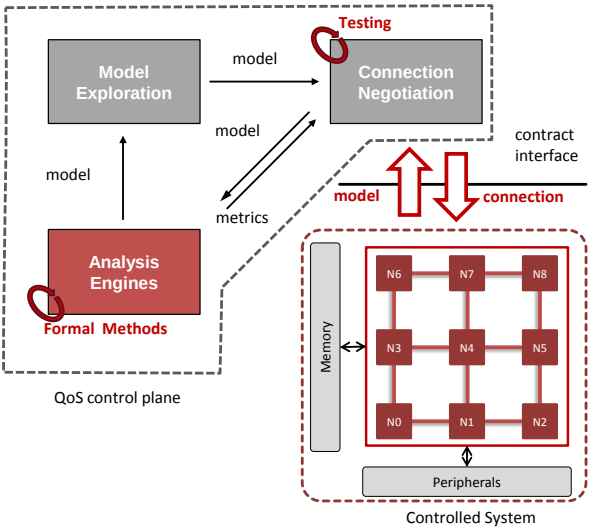


Figure 3.10: Model domain architecture for adaptive arbitration in a NoC.

layer and later enforced with the data layer using the aforementioned mechanisms for predictable resource reservations.

As one of the main goals of this work is to provide a deterministic response time for the senders, the control layer must offer predictable resource allocation which can be verified during the design phase. This requires:

- avoiding contention in (NoC) buffers,
- dividing bandwidth between interfering senders according to their requirements,
- adjusting the settings during the runtime to cope with the dynamics in the behavior of the system.

In the most of existing NoCs, *predictable resource reservations* are controlled through data layer introducing:

- admission control done locally in nodes, e.g., traffic limiters, performance counters,
- arbiters deciding about the allocation of resources to streams in routers,

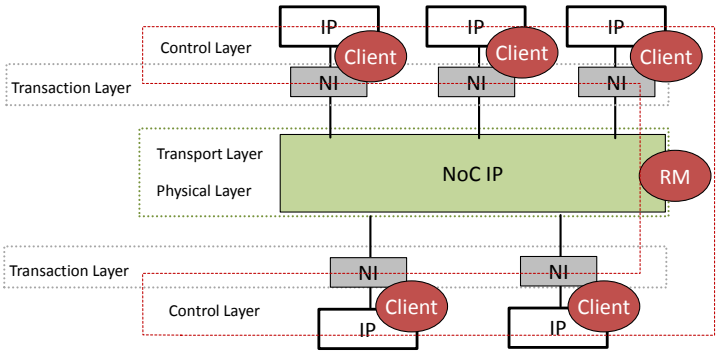


Figure 3.11: New modules introduced by the control layer forming the session layer of NoC communication.

- mixed-solution combining both methods mentioned above.

Based on these mechanisms, we show, in the following parts of this chapter, how a control layer can be applied to achieve simultaneously efficient performance and real-time guarantees in the presence of dynamics. The proposed scheme opens multiple implementation possibilities. The data layer encompassing independent, local resource schedulers must be implemented on each resource in the system which can be accessed by more than one communication participant (processing node/tile). However, for the control layer, there are several negotiation schemes possible since it is orthogonal to the underlying system and may be implemented independently.

3.4 Overview of the Architecture

The dynamic resource allocation requires consideration of the local state of the core (number and execution profiles of currently active applications defining core's requirements for NoC) as well as the global state of the interconnect (number and profiles of active interfering senders determining possible interference from interconnect). Although evaluation of the former "on-core" factor can be done locally (disregarding information about other senders), information about the latter "off-core" factors requires further, global synchronization done by the QoS control plane. The essence of the centralized implementation of the QoS control plane is a resource management (RM) unit and clients.

Figure 3.11 presents an overview of the design which, from a technical perspective, introduces *connection oriented network services* [136] adjusted for providing a predictable network behavior. To use a connection-oriented network service, the communication participant (e.g., IP) must first establish a connection with the **Resource Manager (RM)** unit, i.e. request allocation of the NoC resources for the particular connection. Only after the resources are assigned to the RM, the sender may conduct transmission (use the resources). Finally, the NoC resources must be released whenever they are no longer necessary. In this context, the primary function of the RM is to establish and manage all aspects of this process, i.e., the RM's arbiter must decide the order and duration of connections (transmissions). However, assuring the predictable behavior of RM is not enough for providing temporal guarantees. Note that malfunctioning or malicious senders may easily influence work or the RM. For instance, wrongly configured applications could send too many requests delaying or blocking the work of the arbitration unit. They could also wrongly/maliciously set their connections, e.g., ask for too many resources (blocking other senders) or occupy them for an extended period (or even never releasing them).

Due to their importance, these problems must be addressed whenever temporal predictability is required. Otherwise, no real-time guarantees can be given. Therefore, the proposed mechanism introduces **clients** - the admission control units (local supervisors) running locally on each processing node in the NoC. Clients are responsible for: i) establishing connections with RM (issuing correct messages to RM for appropriate actions of senders and processing node); ii) preventing non-authorized accesses to the NoC iii) adjusting local admission setting in NI, e.g., rate settings for rate limiters or MMU/MPU address translation tables, based on the configuration messages from RM; iv) prevent non-authorized or too frequent accesses; v) releasing the NoC resources (informing the RM whenever an application terminates), and vi) preventing unbounded NoC accesses (release the NoC resources and notify the RM whenever a connection takes too long and/or terminates too long connections).

Figure 3.11 presents the client and RM, forming the session layer (in red color). The client modules can be implemented as hardware extensions of the NI (for high performance) or as a software module running on the IP core (however also in this case some extension of the NI may be necessary). Similarly, RM can be fully realized in hardware, i.e., as an independent HW IP connected to the NoC, or as a software IP running on one of the nodes. The main advantage of the software realization is flexibility. The complexity of both units depends on the complexity of the selected synchronization protocol and will be discussed in the next sections.

Separating the clients and RMs from software running on processing nodes is

especially important in safety-critical setups where a NoC is treated as a shared resource. Recall, that in such setups it is necessary to either certify the whole system (including all applications) to the highest relevant safety level or decouple the resource arbitration from senders for providing sufficient independence [67], [5]. As assuring adherence to standards is a costly and demanding process, the latter is the preferred solution in most setups. This can be achieved through clients and RM which can be certified/designed independently to the highest relevant safety level.

3.5 Synchronization with the Control Layer

In general, the process of global synchronization can be divided into the following phases: *initialization*, *reservation*, *usage*, *release*. Fig. 3.12 presents the method in the context of a motivational example: the application (sender) issues a request to the RM for providing access to a DDR-SDRAM. Firstly, the client must detect the access from the application (IP) and block (trap) it until RM grants the connection. Next, the client must issue a request to the RM to establish a proper connection, Fig. 3.12.a). After receiving, evaluating and processing the request, the RM can grant the access to the resource - DDR-SDRAM memory, Fig. 3.12.b). Upon arrival of the grant message from the RM, the client unblocks the sender, and the connection may become active, Fig. 3.12.c). The Sender might have access to the resources for a predefined amount of time or length of the particular transmission (e.g., number of issued packets), depending on the actual implementation. After the transmission finishes or the connection length has been reached, the client must terminate the communication and release the resources (and eventually block the sender). This is done with the appropriate release message issued to the RM, Fig. 3.12.d). The resources are considered to be free again after the RM has processed the release message.

Now, the process of synchronization will be discussed in detail. This description will be used for presenting different versions of the protocol depending on the temporal requirements of senders.

Fig. 3.13 provides a generalized version of the synchronization workflow. Note that the actual steps and their details may vary according to the selected protocol and implementation. In the figure, a sender conducts an access to a virtual resource composed of two real hardware resources, i.e., connection (e.g., a path through a NoC with a predefined QoS) and the exclusive access to a selected hardware module (e.g., DDR-SDRAM controller). To satisfy this request, the RM firstly evaluates the available resources while considering the global state of the system.

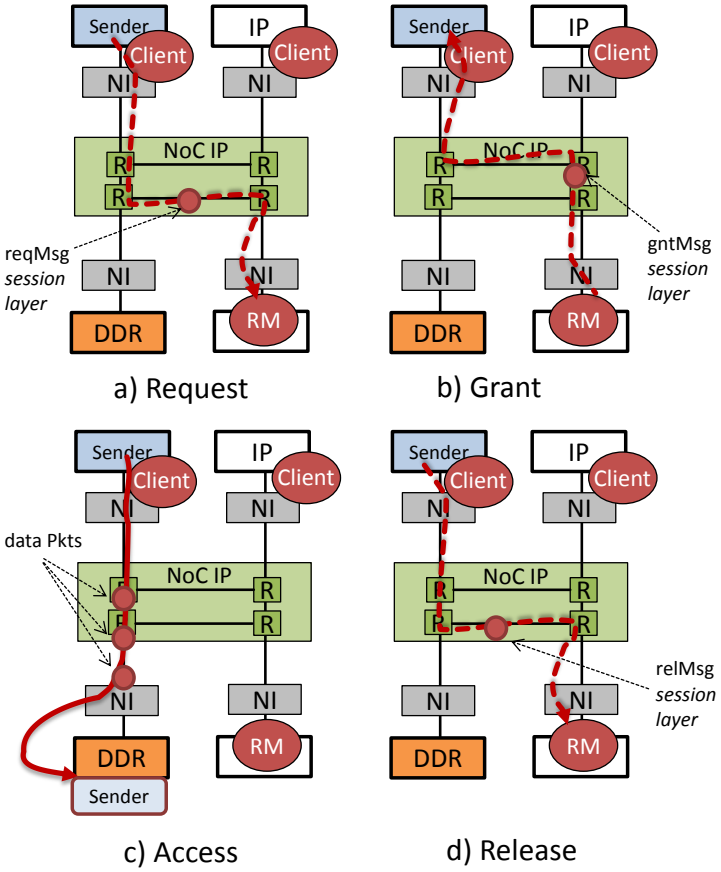


Figure 3.12: New modules introduced by the control layer forming the session layer of NoC communication.

This is done based on the defined system model and the selected arbitration policy. Next, the RM locks NoC resources for the requesting sender and (if necessary) re-configures the settings of the NoC (e.g., traffic shapers/rate limiters in egress ports of NIs) to fulfill the requested connection parameters. The primary goal is to assure that every requesting sender receives the maximum available service for the given state of the system, as well as that the transitions between system states (i.e., mode changes) are safe and reliable, i.e., avoid sporadic overloads. After the trans-

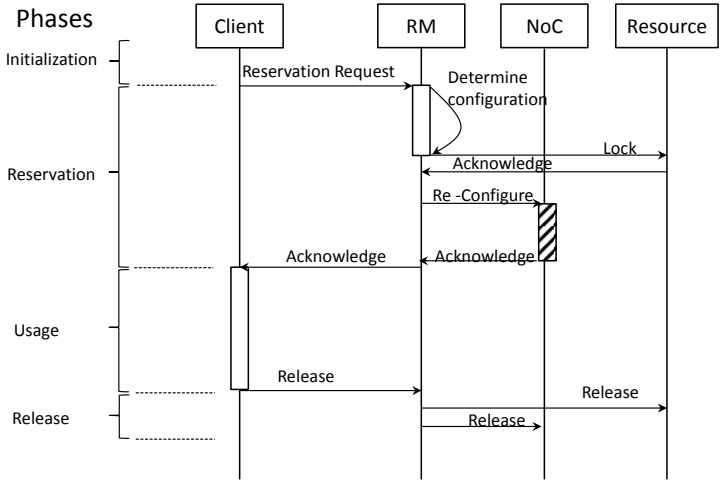


Figure 3.13: Exemplary workflow of the control layer specifying synchronization phases.

mission is complete resources must be once again released for other senders. The complexity of the protocol, i.e., each of the reservation phases, depends directly on the resource allocation scheme applied by the control layer (designer) as well as on the concrete architectural features of the selected, underlying system architecture.

The following sections will provide an overview of the necessary actions, covering all aspects of these phases in the scope of conducted operations and participating elements of the system. Possible implementation methods and required tools will be detailed in Chapter 4.

3.5.1 Phase One: Initialization

First, the access from the synchronized sender must be detected and evaluated to provide information about the QoS requirements for the requested connection. Later, this information must be forwarded to the central scheduling unit - the Resource Manager.

The prerequisite is HW support for the admission control, i.e., NIs/Tiles should support at least monitoring for rate control. This assumption is realistic as rate limiters are provided by the majority (if not all) of contemporary MPSoCs, e.g., R-Car [120], Cell [115], KiloNoC [57], IDAMC [140], MPPA [41]. The considered baseline NoC architecture also supports them. The main change introduced by the control

layer is the need to re-configure the rate settings at runtime. The values of rate-limiters are currently set only once during the boot-up phase of the SoC or hard-coded in the modules. Consequently, adjusting them at runtime may require an additional extension of the interface. An MMU or MPU would be required if the NoC should provide fine-granular protection, e.g., distinguish between different address ranges. Both can be found in most of the new controllers, such as the R-Car from Renesans but also in the Cell processor. Therefore, the synchronization can be initiated by:

- NI through monitors (Rate Ctrl + MPU/MMU),
- Tile with Rate Control (implemented in HW or OS-SW).

The accuracy/granularity of synchronization constitutes the major design trade-off. In the first scenario, a monitor in the NI can recognize only the addresses accessed by the tasks running on the tile. As a result, distinguishing between specific senders may be difficult. Therefore, synchronization can be applied either for the whole tile (e.g., monitoring of joint workload resulting from traffic initiated by a group of tasks) or for the specific address ranges (whenever the NI supports MPU/MMU protection). The second scenario considers setups where the tile provides additional support for identifying senders and processing messages from the RM. This could be done by extensions of SW (e.g., dedicated syscalls) or HW (e.g., interrupts) IPs. At the cost of this additional resource overhead, it is possible to identify specific tasks which are running on tile, or higher granularity, particular actions conducted by applications (e.g., concrete DMA transfers).

Similarly, the clients logic (responsible for synchronization with RM) can be developed as SW or HW IP. The software deployment decreases the size of the NI. It requires only an extra interface for programming rate limiters and MMU/MPU. On the other hand, the HW implementation improves performance. Complexity and goals of clients depend on the resources which are available for implementation as well as characteristics of the running senders and therefore vary between setups. Independently of the method of deployment (HW or SW), the actions of the client are transparent to the sender.

3.5.2 Phase Two: Reservation

Each request must pass several processing steps defining the internal working of the RM. These steps are presented in Fig. 3.14. Firstly, the new request is evaluated concerning the availability of resources depending on the current state of the NoC, i.e., the number of running senders and conducted transmissions. Later, the

RM must check if the requested connection can be realized in the current system configuration assuming the selected arbitration method, i.e., conduct an *admissibility test*. For this purpose, the RM requires an up-to-date model of the NoC and a set of rules - the arbitration policy. The former model describes the form of a data structures with information about resources, e.g., links and buffers in NoC, their capacity and utilization. The primary goal is to check which NoC resources are occupied by senders and what are the QoS requirements of currently running transmissions. The arbitration policy must decide the order of allocation of requests. For instance, all requests can be treated equally (round-robin policy) or a priority based preemptive or not-preemptive schedules can be used as well as time-based allocations (e.g., time-division multiple accesses TDMA). Additionally, RM may introduce advanced arbitration policies oriented on selected properties of the system. As an example, the RM arbiter can optimize the utilization of DDR-SDRAM by protecting locality of memory accesses. In this case, the arbitration must demand that:

- only one transmission from the synchronization scenario can be active at a time,
- the change of an active transmission is possible only after the previous one has finished or it is preempted due to a transmission request with a higher priority,
- the RM must prevent starvation of requestors.

If there are enough resources to handle the request, the RM may allocate them to the requester and notify the client (or sender) about the success. If this is not the case, the management unit may look for another possible setup/configuration, e.g., adjust properties of other clients so that the requirements of all senders can be met. If reservation requests are provided with a priority-based arbitration, then the RM must enforce the proper order of re-configurations during the allocation phase. This can be done in both preemptive or non-preemptive manners. If the former method is selected, the RM can deprive senders with lower priorities from resources as soon as the hi-priority request arrives. This operation inherently causes a change of the system mode and as a result may cause a sporadic overload situation which endangers the system's safety [105],[78]. For instance, although the RM may send a control message to block individual nodes, the flits/packets injected before the arrival of this message must still leave the NoC and could interfere with higher priority senders. Therefore, such effects must be accounted for during the design of control protocol to assure safe and efficient mode changes. A detailed description follows in Section 3.6 and Chapter 4. If a non-preemptive method is selected,

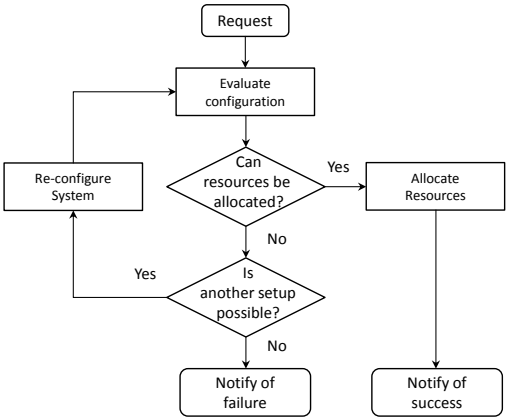


Figure 3.14: Steps required for the re-configuration of QoS control plane and its operations in NoC domain.

the higher priority request must wait for an ongoing lower priority transmission to finish, but this blocking happens only once.

In situations when there are not enough NoC resources to handle the request, the RM may actively reject the transmission or inform the requester about the foreseen length of the blocking, see Section 3.7. This creates an interface between on-core and resource scheduling, e.g., on-core preemptions. Similarly, the RM may re-order (or prioritize) requests based on the properties of resources which they are targeting, e.g., state-dependent nature of DDR-SDRAM scheduler. Details on the interfaces to on-core and resources scheduler are provided in Sections 3.7 and 3.8. Finally, the RM must communicate its decisions (appropriate regulator settings) to all the involved senders/clients with control messages.

3.5.3 Phase Three: Usage

The resource usage session starts directly after the client receives an acknowledge of a request. The client modifies/adjusts the regulator settings *only* after receiving the settings from the RM depending on the current *system mode* communicated by the RM. Furthermore, the acknowledgment may allow:

- *entire transmissions constructed from multiple packets* for instance a DMA transfer,

- accesses to the NoC with a pre-defined rate (settings for rate-controllers) for cache based traffic.

The settings during the usage phase may change at runtime. Firstly, the application may be blocked to allow transmissions with higher priorities to progress with their execution. Secondly, the allowed access rate may be increased or decreased depending on the global state of the system. In case of cache lines, this effectively throttles (accelerates or slows down) on-core execution of senders.

Finally, for some applications the settings may be pre-defined (during the design phase) and constant during the runtime, i.e., they stay for the whole execution duration in the usage phase. For instance, the designer may decide to keep the rates for some applications at a constant level (without endangering their deadlines) in some or all modes, to limit the number of necessary synchronizations and reconfigurations. Additionally, the fully dynamic reconfiguration may be conducted only for selected senders or usage scenarios, e.g., fail-over recovery assigning more bandwidth to a select sender or data flow in case of a need for re-transmission caused by a transient error.

3.5.4 Phase Four: Release

The mechanisms require that the resources are released by clients whenever a sender does not use them. The releases introduce dynamics which improves not only the utilization of the system but also, even more importantly, enables better formal guarantees in most setups.

The release method strictly depends on the requesting procedure and has similar advantages or drawbacks. Apart from release being done at the end of transmission, e.g., signalized by the last packet, flit or interrupt, the release may be enforced by a monitor after a predefined timeout. This is done to assure a safe upper-bound on the resource usage sessions, i.e., avoid infinite accesses which could be caused by malicious or malfunctioning applications.

Summary of Phases Description

The mechanism provides a QoS abstraction of the underlying NoC (data plane) allowing a path oriented approach in which both the per-hop behaviors of routers and the end to end properties of communication can be unified. This unification enables safe and efficient resource reservations but requires knowledge about the global state of the system, i.e., number of simultaneously running senders, their current state and QoS requirements which may change during runtime. Therefore, for establishing and adaptively managing connections, clients must negotiate

reservations. The negotiation is done through an exchange of messages between nodes with interfering senders. The communication is forming a synchronization protocol with the purpose of propagating information about the state of a connection as well as current QoS requirements for a particular sender. Note, that the latter parameter can change dynamically and the contract based negotiation safely adjusts the QoS settings of the platform.

Several negotiation schemes are possible with two pre-dominant patterns: direct communication between clients or communication through a single scheduling unit (Resource Manager). This problem translates to the classic synchronization dilemma differentiating between centralized and decentralized resource assignment scheme. Trade-offs between both approaches are well researched and described in the existing literature, e.g., [74], [103], [14]. Application of one from these schemes is usually setup dependent and influenced by the number of synchronized senders, their mapping, parameters of the transmissions (length, duration, etc.) as well as particular architecture (e.g., propagation latency of the single control message). In the next section, a couple of exemplary resource arbitration protocols will be used to familiarize the reader with the mechanism and present the possible trade-offs.

However, the main advantage of the mechanism, common to both synchronization schemes, is that it makes the QoS functions in the routers agnostic to achieving the real-time and safety guarantees i.e., these functions can be incorporated to existing NoC architectures by adjusting admission control in NIs with the proposed mechanism. The admission control and adaptive management of NoC state are entirely controlled by the introduced QoS control plane making the system potentially more efficient. The resource arbitration can be optimized and adjusted to the changing global state of the system for accommodating arriving workloads as well as reacting to possible errors.

3.6 Synchronization Protocols

The important advantage resulting from the decoupling of the admission control from the underlying NoC infrastructure is the broad spectrum of the synchronization protocols (allocation schemes) which can be implemented with the control layer. Recall, that the control layer may implement non-work conserving (e.g., time-division multiplexing) and work-conserving arbiters (e.g., round-robin arbitration) covering the majority of the sharing schemes known from the literature e.g. [92], [121] Moreover, it is also possible to use custom arbiters for particular deployments i.e. adaptive arbitration.

This flexibility allows extending the capabilities of a selected NoC-based platform without the need modifications of routers. For instance, the introduced solution permits the implementation of a TDM-based arbitration on top of a NoC with the performance-oriented arbiters in routers, e.g., iSLIP arbiter [101]. Consequently, the control layer permits increasing the spectrum of system applications, e.g., adjusting it to provide guarantees in real-time domains, without a need for re-design or costly hardware extensions.

The following section will describe several resource allocation strategies (and synchronization protocols) for the centralized implementation of the mechanism. Although a centralized point of synchronization through a single RM unit may seem to be a performance bottleneck, it also has significant advantages when it comes to safety and formal verification in case of SDNs, see [137]. Firstly, it simplifies the synchronization protocol, e.g., (rapid) spanning tree protocol or shortest path bridging. In contrast, decentralized systems frequently require long and complicated protocols for assuring system coherence. For example, in the decentralized scenario, assuring that all senders agree on the global state of the network before they can take actions, significantly increases the complexity and volume of communication in larger scenarios. Moreover, simpler clients and RM resulting from the single point of synchronization decrease the costs of verification and certification which play a critical role in many real-time applications, e.g., avionic and automotive domains.

Note, that a centralized synchronization is inherently suffering from the scalability problem. Therefore, a possible future extension of the presented work could consider systems with multiple RMs controlling different regions of the NoC. Moreover, it would be interesting to investigate fault-tolerant mechanisms, as a further development of this thesis. These mechanisms would explore protocols for the control layer capable of performing design-space exploration within the system model to find a feasible solution for resource allocations. This would be done whenever the requested contract could not be established as well as for adjusting the NoC settings, e.g., robustness against faults of the components. Consequently, in the future, the mechanism may introduce full or partial adaptivity to the architecture.

This section presents multiple examples of synchronization protocols which support different allocation schemes. Note that, the protocols are adjusted for sharing of the same VC between transmissions while preserving service guarantees. This also includes senders with different criticality levels, cf. Chapter 1. This feature of the mechanism permits decreasing the number of required VCs in a system or increasing the number of hosted applications. Additionally, the majority of the proposed protocols derives their efficiency from work-conserving scheduling

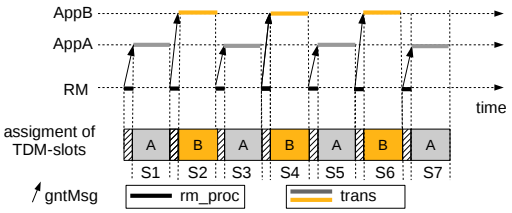


Figure 3.15: Workflow of the TDM-cyclus enforced using RM in a NoC with protocol overhead (rm proc) and durations of two different transmission (trans).

which tries to keep interconnect resources busy for a maximum period of time.

3.6.1 Time-Driven Scheduling

Static isolation is a popular method for managing contention in a NoC as tt provides service guarantees that are independent of other tasks' behavior, cf. [92], [121]. Consequently, each task can occupy the interconnect for a given, predefined amount of time. After entirely using its time budget, the sender must release the resource and allow the next task to proceed with the transmission. The budgets are replenished in cyclic order. The proposed control layer can implement both popular variants of time-driven scheduling: *time-division multiple access (TDMA)* and *round robin (RR)*.

Time-Division Multiple Access

As discussed in Chapter 2, the time-division multiplexing (TDM) approach constitutes the commonly applied solution for real-time systems and is a dominant standard in the avionic domain. According to this scheme each application is granted a dedicated time slot during which it acquires exclusive access to the interconnect. Transmissions are performed in a cyclic, static order forming a TDM cycle. Consequently, the designer may enforce full temporal isolation between independent transmissions, i.e., guarantees for running applications are independent of activities of other senders running in the system.

Fig. 3.15 presents the workflow of such solution. In the considered example, two senders (A and B) form a TDM-cycle in which they acquire access to the NoC alternately. The RM sends the messages - *gntMsg*- granting access for a selected core for a predefined amount of time[2] - its time slot (slots S1-S8). The time necessary

[2]Note that NIs must have the ability to measure time (e.g., a time window for rate limiters).

for the RM to generate the message and its propagation latency in the NoC constitute the overhead of the scheme when compared to the traditional TDM deployments done in hardware. However, the main advantage of the proposed solution is constituted by its applicability to most of commercially available NoCs, such as MPPA or Tile64. The proposed RM-based TDM admission control can be introduced "on-top" of existing, commonly used wormhole-switched NoCs with multistage arbitration. In contrast, existing NoCs supporting time driven scheduling require custom, and frequently complex, router architectures[3], e.g., PhaseNoC [116], SurfNoC [150], Aethereal [56]. Moreover, TDM scheme, when implemented with the control layer, can be easily extended to incorporate elements of round-robin work-conserving scheduling e.g. [80].

Round-Robin Based Performance Optimized Arbitration

Although TDM-based resource allocation protocol allows an easy implementation and provides timing guarantees, it also results in average latencies which are very close to the worst case even when the system is not highly loaded [59]. This is mainly due to the traffic from general-purpose applications that hardly ever follows a constant, predictable pattern assumed by the TDM approach.

The RM-based control layer helps to overcome this challenge as it permits to implement the round-robin based resource allocation, well known from the real-time literature [92]. The presented description of the protocol with the round-robin scheme is based on [81], [82]. Similarly to the TDM-based design of the control layer, the interconnect is considered to be the global resource shared in time and transmissions are granted access to NoC in the cyclic-repetitive order. However, the slots of non-active senders can be released (re-allocated) to serve pending requests from active senders faster, i.e., therefore introduced arbitration policy is work conserving.

The round-robin scheduler avoids unnecessary idle times and offers a more compact schedule, i.e., it tries to keep the NoC resources busy whenever there are pending transmissions. Consequently, a round-robin based control layer improves average performance without decreasing safety guarantees, i.e., it offers the same worst-case performance/guarantees as TDM.

Workflow The communication protocol for a round-robin based allocation of resources is realized using three control messages: *reqMsg* (request), *relMsg* (release) and *ackMsg* (acknowledge), cf. the workflow in Fig. 3.16. The corresponding client

However, also a setup is possible where only RM has the timer. In this case, RM must firstly block-/preempt currently ongoing transmission from the previous slot and later send gntMsg, as described in the Sections.

[3] Please, refer to Chapter 2 for detailed discussion.

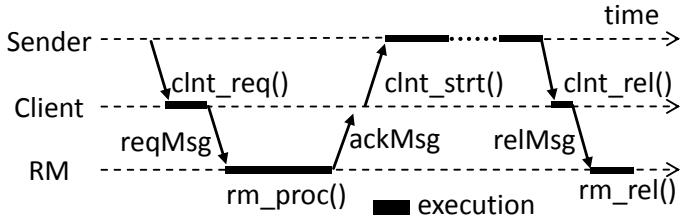


Figure 3.16: Workflow of the RR-based admission control in a NoC implemented with the control layer, based on [81].

sends a request message to the RM (*clnt_req*()). The RM is equipped with a queue for storing pending requests from clients. If the queue is empty, the RM must wait for a new request to arrive. Otherwise, the scheduler decides which request from the queue should be served first (*rm_proc*()). The accesses are granted in the predefined cyclic order, but unused slots (for which there are no pending requests) are skipped. The *RM* must notify the sender (which is selected for service) with the *ackMsg*. After receiving the *ackMsg*, the communication may start (*clnt_strt*()). Once granted, the connection holds until the end of the transmission or the abortion through the client based on a predefined timeout used to prevent unbounded connection latencies. When the client detects the end of the transmission (e.g. based on its time-budget or injection of the last flit) it issues a *relMsg* to the RM (*clnt_rel*()). As soon as the *relMsg* arrives, the RM considers the resource to be free again (*rm_rel*()).

RR-based Control Layer vs TDM The proposed solution overcomes the main drawbacks of the TDM based arbitration. Firstly, due to the work-conserving arbitration, the introduced control layer decreases average latencies, i.e., temporal over-provisioning, of the applications which are not optimized for the TDM-cycle. As presented in Fig. 3.17.a), in a system with a TDM based arbitration, whenever tasks expose dynamics in their behavior, e.g., even with a small jitter, transmissions are blocked, and their execution is delayed for the duration of a whole TDM cycle. In contrast, when the RR-based arbitration and the control layer are applied, transmissions can be scheduled whenever the NoC is free, see Fig. 3.17.b). This improved arbitration is possible due to the introduced arbiter which tries to process the requests as soon as they arrive and re-allocates the unused slots. Whenever the interconnect is not heavily loaded, see Figure 3.18 the unused slots can be omitted (re-allocated) for pending requests. This re-allocation is especially useful for longer transmissions composed of multiple packets, e.g., DMA transfers which can progress faster on average without idling between consecutive TDM cycles.

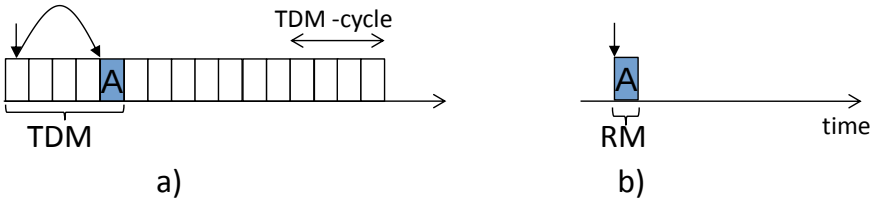


Figure 3.17: Incorporation of dynamics in activation patterns in a system with a) TDM-based arbitration and b) RR-based control layer.

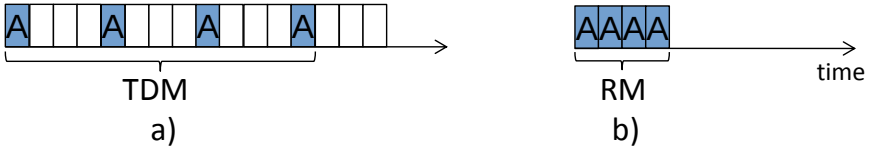


Figure 3.18: Dependency between blocking and the actual load of the system for a) TDM-based arbitration and b) RR-based control layer.

Furthermore, the arbitration of large scenarios with multiple senders exposing dynamics in their activation patterns (e.g., modes of work, interrupt signals) is done without the need for complicated and hard to maintain TDM-cycles. The proposed mechanism maintains the locality of memory accesses [113] through the isolation of the whole transmission.

3.6.2 Static Priority Based Arbitration

This section presents an alternative approach towards assuring QoS in the NoC. The control layer is adjusted to provide arbitration in mixed-critical real-time systems. In such setups, different requirements of senders (regarding NoC resources and/or different importance for the safety) are reflected with priorities assigned to the transmissions e.g. deadline monotonic priority assignment. Consequently, the arbiter used by the RM must adjust resource reservations depending on the importance (priority) of the request. The following of this section describes the static priority based RM protocol, according to which resources are always granted to the request with the highest priority. The description is based on [83] and [75].

The synchronization between clients and the RM is accomplished using five control messages: *reqMsg* (request), *relMsg* (release), *ackMsg* (acknowledge), *blckMsg*

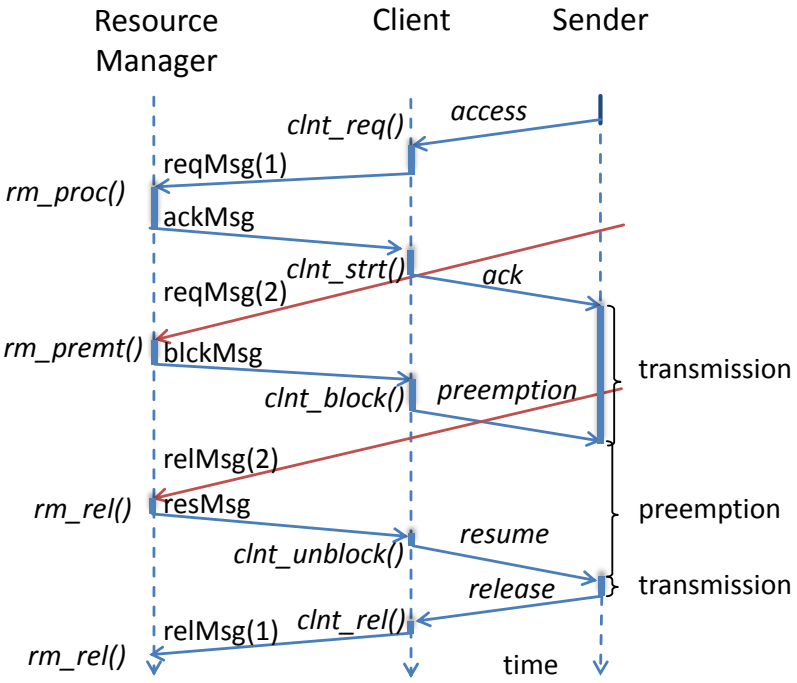


Figure 3.19: Workflow of the SPP-based synchronization protocol in a NoC utilizing the control layer, based on [83].

(preempt communication) and *resMsg* (resume communication). The workflow is depicted in Figure 3.19.

Clients issue a request message *clnt_req* whenever supervised sender is trying to access resources. If there is no pending access, the *RM* must wait for a new request to arrive. If a request cannot be granted, due to an ongoing higher-priority transmission, it is stored in a request queue. When the resource is free again, and the request queue is not empty, *RM* selects the request with the highest priority. The *RM* notifies the appropriate client with the *ackMsg* which unblocks the granted transmission. If a resource is occupied, i.e., there is an ongoing transmission, the *RM* must monitor all arriving requests and compare their priorities against the priority of the currently ongoing transmission. Comparison and monitoring are done in the arrival order. If the priority of the request is lower than the priority of the ongoing transmission it is stored in the queue; otherwise, the

RM conducts a preemption. In order to preempt an ongoing transmissions, the *RM* sends a *blckMsg* to the client supervising it. Preemptions may be nested, i.e., transmissions which previously preempted an ongoing communication can also be preempted. Therefore, a *RM* must store information about ongoing and preempted transmissions. Moreover, the RM sends the *ackMsg* with a delay to ensure that packets from previously ongoing transmission have left the NoC, i.e., provide logical isolation. After receiving the *blckMsg*, client blocks the next packet from the ongoing transmission as soon as possible. The preemption granularity depends on the admission control in NI (rate limiter). In this work, assuming the baseline architecture, the arbitration is performed at the packet level. Because preemptions are happening on the same VC, the flit level is unacceptable due to the properties of wormhole routing (cf. [37]). Higher preemption granularity, e.g. multiple packets to ensure the locality of a transmission, is also possible after accounting for an additional latency. When the client detects the end of a transmission (e.g. based on its time-budget/timeout or injection of the last flit), it issues a *relMsg* to the appropriate *RM*. As soon as the *relMsg* arrives, the RM considers the resource to be free again. If there is a pending preempted transmission with a lower priority, the *RM* resumes its execution after sending a *resMsg* to the appropriate client. Otherwise, a new request is selected from the queue. As soon as the *resMsg* arrives, the client unblocks corresponding preempted transmissions.

RM-based SPP vs SoA Solutions. As discussed in Chapter 2, prioritization of traffic conducted locally in router is a popular solution for allocation of resources in NoCs e.g. [31],[66]. The main disadvantage of this scheme is the high resource overhead (demand), i.e., streams of different priorities must be isolated in separate traffic queues. Consequently, the number of VCs must be equal to the number of criticality levels in the system and therefore must increase accordingly. The constant increase in the number of applications integrated into a single chip, e.g., Flight Management System [46], requires the number of VCs to be equal to the number of criticality levels and to increase accordingly, otherwise the system is not predictable [55]. The proposed scheme based on the control layer offers efficient sharing of the same buffer queues in routers while preserving the priority based arbitration. This feature is especially important in mixed-critical setups.

Additionally, formal verification of priority based routers can be complicated. For instance, effects of backpressure (see Chapters 1 and 2) could lead to the propagation of blocking and cyclic dependencies between streams. The proposed protocol for the control layer avoids these problems using the global arbitration scheme. The blocking is moved from the NoC to the cores, therefore i) it is possible to reduce the size of buffers significantly [83] ii) one could re-use the blocking time on-core for execution of other tasks, which is described in detail in Section 3.7.

The control layer guarantees also locality of memory accesses, similarly to RR- and TDM-based protocols. This is not possible in case of priority based scheduling which is performed locally in routers.

3.6.3 Dynamic Priority Based Arbitration

The strict priority-based scheduling done locally in routers decreases the performance of best effort senders. These usually receive the lowest possible priority [43],[83], and thus they are scheduled only when there is no other pending transmission. Consequently, BE senders suffer from high latencies and jitter.

Slack-based resource allocation is a well-known solution from the scheduling theory [92], [39] to this integration challenge, providing high performance for best-effort (BE) and soft real-time transmissions (SRT'Ts) without violating the timing constraints of hard real-time transmissions (HRT'Ts). According to this approach, BEs and SRT'Ts are scheduled whenever the execution of HRT'Ts can be safely postponed without causing missed deadlines. This additional delay is possible whenever a slack is available, which is the time budget between the worst-case response time of a hard real-time application and its deadline. Applying this principle to NoC significantly increases the performance of soft real-time and best effort data streams which is particularly useful for general-purpose latency sensitive applications running on processors with caches [144, 141, 145].

This section proposes an extension of the previously introduced priority-based protocol for the control layer. The description is based on [75]. Consequently, the RM safely *delays* the execution of HRT'Ts, whenever it is possible, in order to improve the performance of SRT'Ts.

A prerequisite of the introduced arbitration is an offline computation of time budget for each HRT'T called *slack* (see Chapter 4, Section 4.1.1), which defines the maximum delay a HRT'T can experience without endangering its deadline. Later, the RM monitors the slack budgets of *currently ongoing* HRT'Ts and dynamically adjusts the priority of SRT'Ts. SRT'Ts get the highest priority whenever there is an available slack and the lowest priority whenever there is no slack. Note that delaying the execution of a HRT'T with the highest priority also delays all currently preempted HRT'Ts with lower priorities. Therefore, this delay can only occur if enough time (i.e., slack) is available for all HRT'Ts to meet their deadlines.

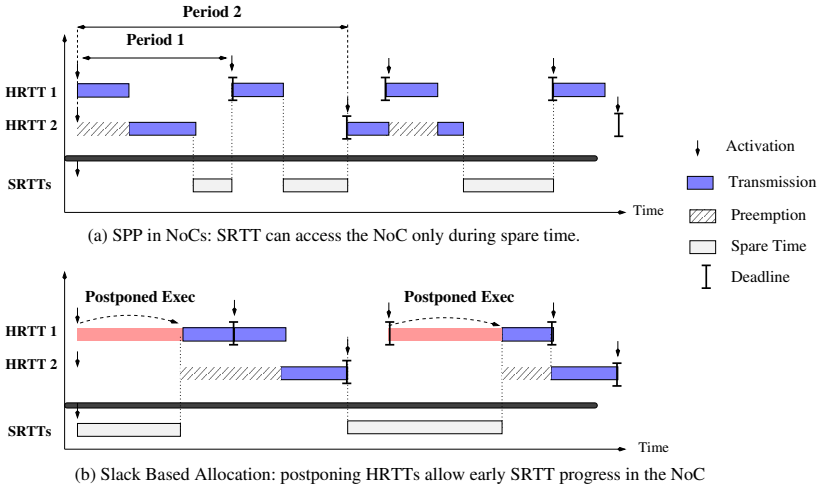
Workflow The implementation of arbitration based on dynamic priorities is realized using five control messages: *reqMsg* (request), *relMsg* (release), *ackMsg* (acknowledge), *preMsg* (preempt) and *resMsg* (resume). Upon arrival of a request message (*reqMsg* to the RM, each request is classified as SRT'T or HRT'T and stored in the appropriate queue. FIFO is used for SRT'Ts and a priority queue for HRT'Ts. As

described previously, the RM monitors all HRTTs and keeps track of the amount of available slack. When a SRTT is granted access to the NoC, the RM decrements in real time the slack budgets of all active pending HRTTs, i.e., it considers all requests in the HRTT queue. The slack budget is renewed at each request arrival (*reqMsg*) from a HRTT. If there are no pending SRTTs or the slack budget of at least one HRTT has been exhausted, HRTTs transmissions regain access to the NoC and are scheduled according to the standard static priority preemptive (SPP) policy. Note that, when SRTTs have access to the NoC, the RM must release the resource early enough to guarantee that HRTTs start on time (i.e. not after the slack budget has been consumed). When the NoC is free, the RM notifies the appropriate client with an *ackMsg* for the pending transmission to start sending data. If the NoC is occupied and the RM receives a new request with a higher priority, the RM must preempt the currently ongoing transmission. The preemption is realized in the same manner as in case of the protocol for arbitration with static priorities, i.e., *preMsg* for blocking the client and *resMsg* for resuming it. Recall, the high-priority sender must start its transmission with a delay (by *ackMsg*) to ensure that packets from previously ongoing transmission are not present in the NoC. Clients confirm finished transmissions by sending the *relMsg* to the RM which removes the corresponding request from the head of the queue. Next, if there is a pending preempted transmission with a lower priority, the *RM* resumes its execution after sending a *resMsg* to the appropriate client. Otherwise, a new request is scheduled.

Comparison against SoA Only a few existing works, e.g. Backsuction mechanism [43, 141, 145], have considered using slack-based resource allocation in the context of NoCs by applying, locally in routers, dynamic prioritization for different virtual channels. However, the main drawback is the high hardware overhead resulting from a custom router design and the high number of virtual channels (i.e., required buffers) corresponding to the number of priority levels in the system. Moreover, these solutions drastically increase NoC complexity as they require to propagate the global state of the NoC to the local arbiters in routers. This propagation is contradictory to the principle of non-blocking routers which requires no correlation between local arbiters and thereby increases the complexity of the worst-case timing analysis.

Consider the example depicted in Figure 3.20, a NoC shared between a set of HRTTs and SRTTs. It constitutes a mixed-critical system where each HRTT has a unique *static* priority (assigned for instance according to a rate-monotonic scheme) and remaining SRTTs have the same lowest priority. Figure 3.20-(a) illustrates the evolution over time of transmissions in a real-time NoC using a static priority preemptive (SPP) policy [31],[66].

The RM permits to safely delay the execution of HRTTs, whenever it is possible,



(a) SPP in NoCs: SRTT can access the NoC only during spare time.

(b) Slack Based Allocation: postponing HRTTs allow early SRTT progress in the NoC

Figure 3.20: Comparison between SPP and slack-based resource allocation in NoCs, based on [75].

to improve the performance of SRTTs. [4] SRTTs get the highest priority whenever there is an available slack and the lowest priority whenever there is no slack. Note that delaying the execution of a HRTT with the highest priority also delays all currently preempted HRTTs with a lower priority. Therefore, the delay can only occur if enough slack is available for all HRTTs to meet their deadlines. This effect is illustrated in Figure 3.20-(b), where HRTTs are postponed and SRTTs may start earlier. Consequently, the introduced control layer manages to drastically reduce the latencies of SRTTs, compared to Figure 3.20-(a), without endangering the deadlines of HRTTs.

Finally, this arbitration does not require custom router design. Indeed, the proposed solution can be applied for enhancing architectures using simpler routers, e.g., in performance-optimized NoCs.

[4] Recall that frequently in real-time systems, SRTTs can progress through the NoC *only* when HRTTs are not sending data. Therefore they are often unnecessarily delayed. In contrast, HRTTs are scheduled as soon as they arrive, although they usually do not profit from faster execution as long as they are guaranteed to finish before their deadline.

3.6.4 Adaptive Path Allocation

In most existing NoCs, guarantees for senders with hard real-time requirements (HRT) are achieved at the cost of decreased BE performance. To assure predictability, designers apply static resource allocation schemes - oblivious routing - where communication paths are solely defined by source and destination and do not change during runtime, see Chapter 2. This allocation policy results in known, well-defined sets of interfering senders and permits the implementation of the strict spatial separation (BE do not share paths with HRT) or temporal isolation (BE senders are blocked whenever HRT are sending) for providing guarantees. However, a static path allocation scheme significantly decreases hardware utilization and performance. Indeed, if oblivious routing is applied, non-uniform traffic patterns can induce large load misbalances giving suboptimal throughput [37], thus resulting in the overly pessimistic worst-case guarantees for HRT and unfulfilled design requirements for BE senders [147].

To fully exploit the multidimensionality of a NoC's topology, this section introduces a protocol for the operation of the control layer which applies different path allocation methods, i.e., taxonomies, depending on sender's criticality. The description is based on [77], [79]. To handle HRT traffic the control layer applies oblivious routing (static path allocation) whereas BE traffic will use multiple paths from the source to the destination - an adaptive load distribution. Whenever NoC resources are free, BE traffic is allowed to use the shortest (optimal) path to its destination, even if it overlaps with links used by critical senders. Upon activation of the HRT sender, the control layer releases NoC resources by redirecting the BE transmissions to an alternative route, i.e., BE senders use detoured paths (non-optimal) only when critical senders are actively using resources. This arbitration reduces the impact of HRT transmissions on the average BE performance and permits a gradual degradation of service, i.e., instead of experiencing full blocking BE senders experience slightly higher latencies on the detoured path for some packets and consequently lower average latencies.

Workflow Ensuring worst-case guarantees requires to identify interfering senders and to provide temporal synchronization of concurrent transmissions. The former can be achieved by statically assigning a set of paths for each BE application with one of the available allocation methods, i.e., the set of possible paths for a BE sender does not change during runtime. In principle, there are no limitations in alternative route selection, besides the rule that if a link in the path is shared with a HRT sender, this sender must be capable of accepting the protocol overhead. However, if all detoured paths are shared with HRTs and all HRTs are active then the BE sender will be blocked as in the case of strict temporal isolation. The latter

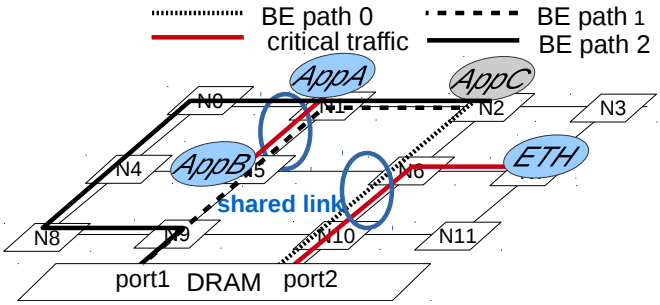


Figure 3.21: An exemplary setup detailing possibilities of spatial isolation in a NoC, based on [77], [79].

condition is achieved through decoupling the admission (the control layer) and flow control (the data layer) in the NoC.

Upon beginning and termination of a transmission from a HRT sender, the RM sends control messages to all interfering with his BEs: *actMsg* (HRT activation), *relMsg* (HRT release). These messages propagate the global NoC state defined by the currently active HRT applications and determine the paths for the BE senders, i.e., after receiving these messages, appropriate paths used by HRTs are blocked or released for BE transmissions. After each mode change, the responsible client must re-program the routing (or address translation) tables in the NI. The changes should force the BE sender to use the shortest path available without any active HRT sender. Finally, to preserve predictability (i.e., isolation) of HRT senders, the RM must account for the delay resulting from the time necessary to deliver control messages to BE senders and for the packets from BE transmissions to leave the interconnect on the selected path. This overhead is acceptable due to the slack of HRT senders which is the time budget between the worst-case transmission latency and its deadline. This method, through a trade-off analysis, provides an estimation of the overhead resulting from the global arbitration, see Chapter 4. Note, that the adaptive path allocation may increase the size of the client and/or NI, as it must store the routing information for alternative communication paths. Alternatively, the RM may send clients routing information in the ackMsg at the cost of increased protocol overhead.

Comparison against static routing. The main advantage offered by the solution is the work-conserving resource allocation scheme allowing improved average performance of the best effort senders. Consider an exemplary setup with the mixed-

critical workload presented in Figure 3.21. As path sharing between BE and HRT may endanger the latter, the traditional safety approach would require traffic from application AppC to take the longest path (*Path* 3). Consequently, AppC will not share any resources (links, buffers) with the HRT sender and spatial isolation will be achieved. However, this decreases the average performance of AppC. It must continuously follow the longer route even when HRT senders are not active.

When the control layer is applied, clients intercept transmission requests from HRT senders (e.g., Ethernet *ETH*0 and AppA) and enforce paths for BE senders accordingly (e.g., AppC) based on the received control messages, i.e., currently active HRT senders. Consequently, if the NoC is free, the BE sender (AppC) is allowed to use the shortest path (*Path* 0) towards DRAM. Upon activation of critical sender *ETH*0 (arrival of Ethernet frame) the RM is sending the control messages (*ackMsg*) to all BE applications sharing resources (links and buffers) with AppC. After receiving this message, the BE sender must redirect its traffic to the longer, detoured path (*Path* 1). If the second critical sender (AppA sending to AppB) is activated the procedure repeats and AppC must take the longest path (*Path* 2). The HRT sender occupies the paths until the end of the transmission/task execution or abortion by the supervisor based on a predefined timeout. When the supervisor detects the end of a HRT transmission (e.g., based on its time budget or injection of the last flit), it issues the relMsg to involved BE senders. As soon as *Path* 0 is free, AppC is allowed to use it once again. As illustrated in Figure 3.21, the detoured transmissions progress faster using the free hardware resources (links and buffers). This enables further exploitation of the NoC's dimensionality.

3.6.5 Adaptive Rate Control

Traffic shaping using rate control is a well-known method for achieving quality-of-service in real-time systems. The goal of this approach is to bound/enforce NoC access patterns and therefore interference which must be resolved through arbiters in NoC routers. The description is based on [78].

As in previously described protocols, tasks/applications must inform the RM whenever they are activated or terminated on processing nodes. Using this information, the RM may decrease or increase the injection rates for a particular node, as depicted in Figure 3.22, dynamically depending on the current system mode. The number of currently active applications defines each mode and determines the minimum time separating every two transmissions issued from the same application.

Workflow The communication is realized with four control messages, see Figure 3.23 including the following operations: activation (*actMsg*), termination (*terMsg*),

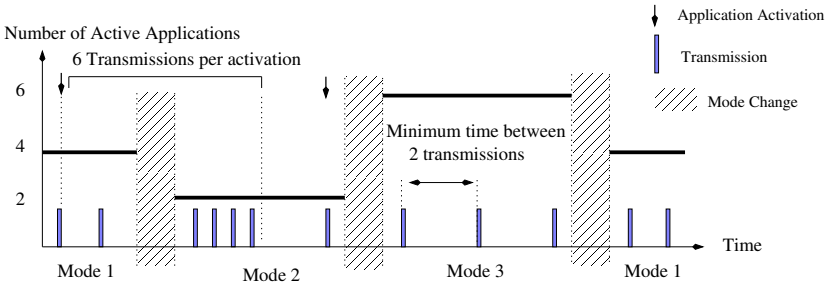


Figure 3.22: A schematic description of the traffic injection rate for a given application, based on [78]. The rate varies according to the system mode.

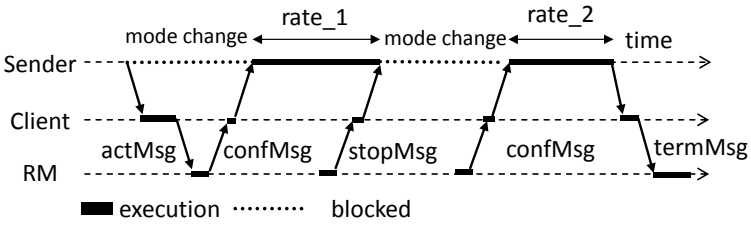


Figure 3.23: Workflow of a protocol for the dynamic rate control with the RM, based on [78].

stop (*stopMsg*) and configuration (*confMsg*). Firstly, the corresponding client sends an activation message to the RM for the activated task on a processing node. The RM processes the activation and termination messages in their arrival order. Each of them initiates the *transition* of the system to a different mode. These transitions are done sequentially, i.e., the new one may start only after the previous reconfiguration is fully completed, for ensuring predictability. Additionally, the RM must assure that the transition between modes is safe. Therefore, before changing the rates, the RM sends, to the clients supervising active applications, a stop message (*stopMsg*) to block all accesses to the NoC from the corresponding node. The clients then wait for the *confMsg* communicating the current system mode. Consequently, the RM sends *confMsg* with a delay to ensure that packets from previously ongoing transmissions have left the NoC, i.e., provide logical isolation. After receiving the *confMsg*, clients adjust the rate and unblock transmissions. Additionally, *confMsg* initiates activated applications (ones which just issued *actMsg*) which can then use the NoC.

Comparison with rate control done locally in nodes. The major drawback of the traditional approach for rate control is that the rates are adjusted statically according to the worst-case scenario, i.e., assuming that all senders are running simultaneously with maximum possible transmission arrival rates. Therefore, this approach is not work conserving and sacrifices the interconnect utilization whenever senders expose dynamics in the execution time, release jitter or communication volume, and the system is not highly loaded. Note that performance penalty increases along with the complexity and inherent dynamics – as in the case of the TDM-based arbitration.

The RM may decrease or increase injection rates for a particular node dynamically depending on the number of concurrently active applications. The mechanism is capable of enforcing symmetric and non-symmetric guarantees. The proposed approach decreases both temporal and hardware overhead. Furthermore, the control layer significantly improves the overall performance of the system and allows meeting the strict timing constraints.

3.7 Interface Between Tiles and NoC

The client forms an interface between the on-core execution of applications and their accesses to the NoC. Therefore, client's functions can be divided into two groups. Firstly, it must control interference which applications running on a processing node may exert on other traffic in the NoC. Secondly, the client must provide the processing node with information about the status of the NoC (i.e., information if this resource is busy or free) to accommodate the new, incoming traffic. The following subsections describe these main functions in detail.

3.7.1 Resource Control in NI

As each processing node is hosting multiple senders requiring different connection settings, clients must distinguish between them. Consequently, connections must be described accurately regarding specific parameters that can be negotiated, e.g., senderId, route (path) through the network, settings of rate limiters and access rights (e.g., read/write transactions). Typically, the sender should produce a flow specification stating the parameters it would like to use. However, in the considered context of the real-time and safety-critical systems, the characteristic of applications with real-time and safety requirements is usually known beforehand and thoroughly tested, cf. Section 3.1.3. These settings are determined during the design phase and encoded/programmed at the chip startup (bootup) in real-time

and safety mechanisms. Examples of such mechanisms are address translation tables and rate limiters which are extending the features of the NI from the baseline architecture.

Therefore, the implementation of clients can be done as an extension of these mechanisms. Figure 3.24 presents an exemplary structure of a client. The described implementation of the control layer follows in a system with a memory-mapped NoC. NI introduces transparent address translation, where the local (for processing node) physical address is converted into a remote address (on another processing core) and forwarded using the NoC. Consequently, senders do not require any knowledge about the operation of the interconnect. The configurable address translation module in the NI does the actual translation (including proper NoC addressing). This module is equipped with tables containing information about routes and destinations (e.g., remote addresses) for initiated connections. The implementation of the client can be done through a straightforward extension of the mechanism. The new columns are added to the address translation table to assess if the synchronization with the RM is necessary and which settings should be applied for ongoing transmissions. These columns must contain the sender ID for identification of the synchronization scenario (assuming multiple of them) and the QoS settings (e.g., a number of packets per base period). Note, that the last field can be omitted at the cost of the higher protocol overhead, i.e., control messages may contain QoS settings, but their length (thus interference) will increase appropriately.

3.7.2 NoC Support for Suspensions

In the majority of safety-critical systems, suspension-based locking protocols, e.g., MPCP, OMLP, FMLP, are used to efficiently and safely coordinate accesses to shared resources. However, existing architectures do not support such arbitration for NoCs although NoCs must resolve conflicts between concurrent transmissions. Enabling suspension-based locking for applications using interconnect resources requires not only predictable latencies of resource operations, e.g., transmission times, but also providing feedback about the global state of the interconnect. This mechanism is relatively simple in classic bus-based architectures, where senders can directly receive feedback about the occupancy of the interconnect from the bus controller using control lines.

However, existing NoCs do not provide such information. In most NoCs the arbitration between transmissions is done independently and locally in each router. Consequently, the state of the network is unknown to the on-core scheduler during runtime, i.e., it assumes the NoC is always ready. Whenever a task accesses the

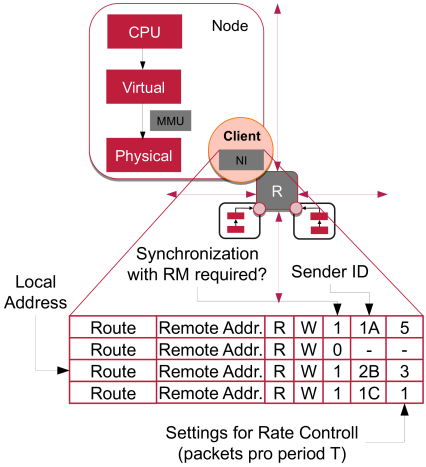


Figure 3.24: Synchronization data for client stored in address translation tables for address mapped NI.

NoC, its processor is stalled (e.g., busy waiting) for the entire duration of a transmission including blocking from other senders. This stalling leads to a decreased processor utilization and overly pessimistic worst-case guarantees as the network blocking propagates to the cores affecting the timing of other (low priority) tasks.

Existing real-time NoCs e.g. [56, 130, 81] can provide an upper bound on the transmission latencies using Quality-of-Service (QoS) mechanisms, such as TDM or dynamic prioritization. However, the decreased processor utilization resulting from the lack of feedback about the state of the interconnect has not been considered until now in the design of on-chip communications.

In commonly used wormhole-switched NoCs with the multi-stage arbitration, ongoing transmissions compete for output ports (link bandwidth) and virtual channels (buffer space). As the arbitration between interfering streams is conducted independently and locally in routers, each network's node has only information about the local status, e.g., information from the flow-control mechanism about the state of the buffers in the neighboring router (back-pressure). Consequently, along with the progress of the transmission through the interconnect it must acquire several resources with independent arbiters and information about the state of the NoC is unavailable. Because of that, some interference cannot be resolved locally by the router's arbiter and requires input from the adjacent neighbors, e.g.,

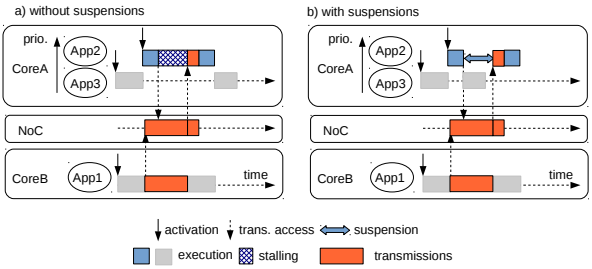


Figure 3.25: Example: Effects of conflicting accesses to the NoC resources from tasks running on different cores in a system with and without support for suspensions.

a joint allocation of the crossbar switch and the router output due to a possible lack of buffers at the output (input-buffered router). Consequently, for the on-core sender, the state of the transmission is unknown during the execution, e.g., it is stalling the core during the blocking.

Figure 3.25.a) depicts an exemplary scenario where three tasks (App1-3) on two cores (CoreA and CoreB) share the path in a NoC with a priority-based arbitration between VCs e.g. [31]. The application's number denotes its priority, i.e., App1 has the highest priority and App3 has the lowest priority. App1 conducts its transmission first. Next, App2 is activated and blocks (i.e., preempts) App3. App2 initiates its transmission, but it is blocked in the routers as its transmission's inherited priority is lower than the priority of the transmission from App1. As App2 does not know how long the blocking will take it is stalling the core. Thus App3 is also blocked and can resume its execution only when both App1 and App2 finish their transmissions.

An alternative (and desired) solution is suspension-based locking which suspends an active application, i.e., App2, waiting for a resource to let other (ready) applications, such as App3, execute even if they have a lower priority, see Fig. 3.25.b). Therefore, the blocking time of App2 remains unchanged but is moved from the NoC to the core. The waiting/stall time may be than used to increase performance and improve guarantees for other tasks. This suspension requires the feedback on the state of the NoC resources (e.g., number of currently active senders and duration of their transmissions) which is not supported by most existing NoC architectures, cf. Chapter 2.

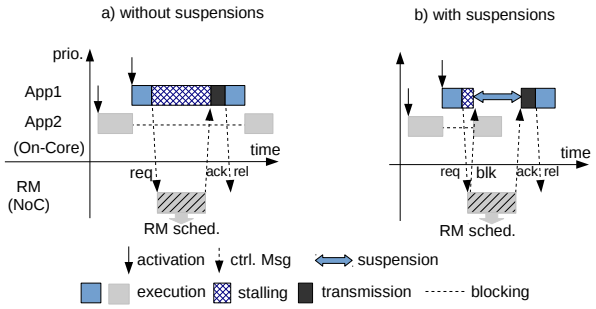


Figure 3.26: Workflow of the RM-based admission control in a NoC without and with the support for suspension-based locking of interconnect resources, based on [85].

3.7.3 Control Layer Support for Suspensions during NoC Accesses

This section presents the extension of the control layer protocol which orchestrates both computation and communication in real-time multi-core systems. The introduced protocol supports suspension-based locking for transmissions from tasks sharing the NoC, i.e., accesses to NoC resources.

The description of this extension is based on [85]. Consider the same example as previously. Figure 3.26-(a) illustrates the evolution over time of the task execution when the control layer is applied to arbitrate between interfering transmissions on the shared NoC. Whenever a scheduled task running on the processor is trying to start a transmission, its request is trapped by the NI/scheduler, and the task is stalled. The corresponding NI/scheduler sends a request message reqM sg to the RM to obtain access to the network. The RM is equipped with a queue for storing pending requests from clients/NIs. Consequently, the RM knows the global state of the interconnect, i.e., which transmissions/tasks are active and which resources (links and buffers) are occupied. The scheduler decides which request from the queue should be served first. After that, the RM must notify the selected application for service with the ackM sg. After receiving the ackMsg, the communication may start. Once granted the access to the NoC, the execution of the stalled task on the processor is resumed, and the connection holds until the end of the transmission. The end of the transmission is signaled with a relMsg issued to the RM. As soon as the relMsg arrives, the RM considers the NoC to be free again.

The RM must have the ability to propagate the information about the state of a particular transmission (e.g. if it is blocked and for how long) to allow suspensions. The mechanism's extension is relatively simple as it requires only introducing a

new block message -msgBlk- to the synchronization protocol. Whenever a request for a particular transmission arrives and the NoC is busy, the RM issues the msgBlk message, as depicted in Figure 3.26- (b), to the requesting node. After receiving this information, the on-core scheduler suspends the requesting task. Similarly, whenever the transmission is re-scheduled by the RM, the arrival of the msgAck must be forwarded to the on-core scheduler.

3.8 Interface Between NoC and Memory

The worst-case timing of running new complex applications in real-time and safety-critical domains, e.g., autonomous driving, is strongly influenced by the latency of accesses to a complex memory hierarchy. Consequently, during the system's runtime applications typically access one or more memory controllers, scratchpad memories, and even one or more levels of cache memories. Some of the elements of the hierarchy are shared only between tasks running on a particular processing node. This work refers to them as local/on-chip memories and for their arbitration one may apply rules known from existing multicore setups, e.g., [27]. However, in MPSoCs selected components of the hierarchy can be jointly used by the plurality of tasks running on different processing nodes. The need for sharing of memories applies especially for high-speed external memories, such as DDR SDRAMs. SDRAMs are often required since on-chip SRAMs cannot provide adequate capacity in a cost-effective manner [10].

3.8.1 Classic Approach for Safe Handling of Accesses to SDRAMs

As already discussed, the primary challenge comes from the need of solving a joint arbitration of multiple resources (on-core schedulers, NoC routers, and memories). However, even SDRAMs themselves constitute a significant problem in safety-critical design due to their stateful nature. Indeed, the latency of a single memory operation depends not only on the amount of interfering requests (e.g., from other senders on different nodes) but also on the commands required to serve them [47]. This dependency makes temporal analysis challenging and results in variable access times making available bandwidth to/from external memory dependent on the traffic history [10].

For instance, given that our NoC admission control enforces the locality of transfers, it is possible to map following physical SDRAM addresses to the same bank/row in a SDRAM device. Therefore, one can employ a contiguous address mapping, as depicted in Figure 3.27. Such mapping decreases memory response time signifi-

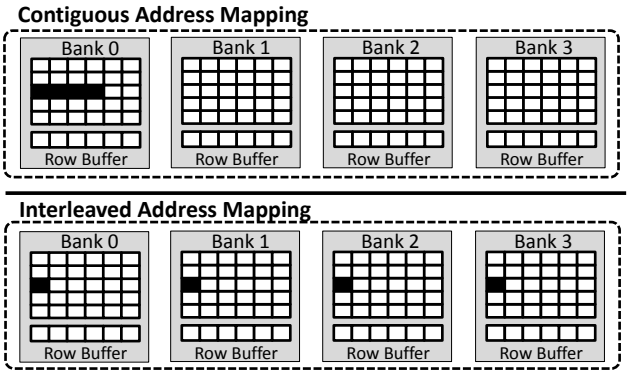


Figure 3.27: Mapping of four consecutive addresses to SDRAM banks in two different mapping mapping configurations (in a system with 4 banks), based on [82].

cantly as consecutive CAS commands that target the same bank row can be executed faster than those that do not. It is so as there is no need for pre-charging and activating a row.

Consequently, the majority of available, performance optimized memory controllers (for instance First-Come First-Served arbitration) are either not sufficiently flexible to manage the NoC's traffic or are hardly analyzable with respect to their temporal properties, e.g., due to sophisticated features, such as support for preemption and reordering [123]. For instance, in NoCs with large TDM slots (that exceed the granularity of the DRAM controller), DRAM locality is also enforced and, consequently, the same standard SDRAM controller can be employed. However, as described in Chapter 2, large TDM slots have the disadvantage of increasing the latency of all requestors in a system, including those who generate non-SDRAM traffic. These shortcomings can be tackled by employing smaller TDM slots or a priority-based arbitration in routers. In such setup, the safe use of a performance-optimized SDRAM controller would require matching the granularity of the SDRAM with the granularity of the NoC. Due to a high granularity of accesses allowing improved scheduling (e.g., contiguous and aligned 8kB long transfers fully benefit from the caching in DDR3) this is either not feasible or drastically decreases utilization of the interconnect.

The alternative would be to map consecutive addresses to different banks, the so-called interleaved address mapping. Bank interleaving is attractive when the locality of incoming accesses is not enforced as it allows to hide the latency of the

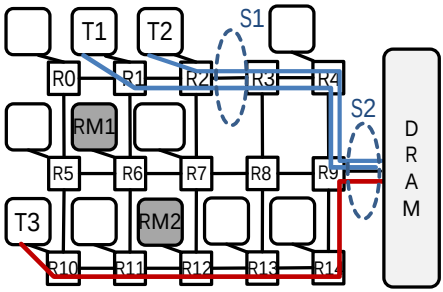


Figure 3.28: Multiple RMs to mitigate the interference on the NoC and memory, based on [82].

row buffer management. Therefore, researchers and designers proposed dedicated memory controllers which are exploiting these effect. An example of non-trivial predictable SDRAM controllers are Dedicated Close Page-Controllers (DCPC) which employ static bank interleaving and closed-page policy (automatically closing the row buffer after using it), e.g., [10]. However, although DCPC makes response time independent of the access history it also significantly increases power consumption [34].

Consequently, the designers in safety-critical domains must confront a hard trade-off between cheap, safe but hardly analyzable performance oriented memory controllers and custom made, expensive and power hungry, dedicated controllers. The next section describes how the introduced control layer mitigates these shortcomings and protects the locality of memory transfers (low memory latency) without a need for custom memory controllers (lower costs and power consumption).

3.8.2 RM-based Admission Control for SDRAMs

The proposed admission control mechanism not only bounds interference on the NoC side, but also on the shared SDRAM side. Indeed, since memory traffic constitutes an essential part of the traffic in a MPSoC system, controlling the interference on the NoC and the memory becomes a primary issue in real-time multicore systems. The proposed admission control mechanism considers a holistic approach, NoC and shared SDRAM, as follows: i) it preserves the locality of memory accesses since the access to the NoC is allocated to the entire transmission, i.e., the requestor fully benefits from the open row policy what optimizes the performance of the system, in addition to ensuring predictability, ii) it mitigates the management of interference between the NoC and the memory controller.

The mechanism is explained using the example in Figure 3.28. Three applications are accessing the shared DRAM memory through the interconnect. Data streams triggered by applications T1 and T2 interfere on the NoC since they are sharing the same path in the network, and therefore form a synchronization scenario S1 managed by the RM1. The data stream triggered by application T3 is using a different path on the network but is sharing the input to the DRAM controller. Therefore, this traffic stream should also be synchronized using the RM2 with the rest of the traffic in the network - synchronization scenario S2. Note that depending on the tolerable synchronization overhead, a *single* RM can be used to synchronize all the traffic in the NoC accessing the DRAM memory. In this case, whenever the RM grants access, a sender acquires exclusive access to both resources: the NoC, as well as, to the SDRAM memory. In this case, the system does not require a predictable memory controller since the RM guarantees temporal isolation of data streams at the NoC and SDRAM level. Note, that the SDRAM controller must me analyzable otherwise it is not possible to provide the worst-case guarantees. However, the control layer simplifies the analysis by limiting the number of possible interference scenarios, and therefore provides more efficient guarantees. This is especially important in case of performance optimized controllers. Consequently, the proposed solution moves the arbitration from resource controllers to the RM allowing joint arbitration without further need for custom hardware extensions.

3.9 Summary

This chapter introduced a novel Quality-of-Service mechanism for NoCs - the control layer. The proposed method uses as a basis the fundamental principles of Software Defined Networks but extends them for real-time NoCs. The control layer decouples the QoS mechanisms from the flow-control conducted locally and interdependently in routers. The QoS is achieved through the online adaptation of admission control in nodes based on contract-based negotiation between senders, i.e., providing a validation method to check if the currently available NoC resources are sufficient for the change in QoS before the application receives physical access to the NoC.

Consequently, routers are relieved of the QoS functions, hence there is no need for custom QoS-oriented NoC extensions. Furthermore, our solution offers the deployment of a sophisticated contract-based QoS provisioning (e.g., round-robin, priority-based arbitration) without introducing complicated and hard to maintain schemes, which are known from the static arbiters. The control layer dynamically adapts the NoC to the changing system's behavior, mode or an environment

which can be influenced by on-chip as well as off-chip factors. Moreover, through the implementation of the arbitration based on the global state of the network (number of running senders and resources used by them), the control layer introduces dynamic interfaces to on-core schedulers as well as peripherals. In the former case, the RM provides information about the state of the transmission (e.g., duration of blocking) which could be used to improve arbitration of local resources, e.g., providing suspension-based scheduling of the CPU time. In the latter case, the protocol allows orchestrating order and duration of accesses to selected shared resources. Such arbitration is particularly crucial in the case of state-based schedulers where the history of accesses decides about latency, e.g., DDR-SDRAM. Consequently, the introduced approach allows achieving safety along with online adaptivity for the high-performance real-time system with dynamic communication requirements. The summary of improvements which can be achieved for the baseline architecture (cf. Sec. 3.1) is available in Table 3.1 and Table 3.2.

Relevant Requirement	Feature \Mechanism Type	Baseline (PS + VC + RL, static priorities)	RM + Baseline NoC (PS + VC + RL, static priorities)
R1	Quality of Support for GL	medium (depends on senders prio.; num. of VCs, and number of prios.)	high (independent of available NoC resources through different allocation schemes may be directly adjusted to requirements of senders)
R1	Quality of Support for GL	medium	high
R6	Performance of BE senders in mixed-critical scenarios	low(high. avg. latencies)	high (possibility to apply slack-based arbitration) low average latencies
R2	Sender Penalty for Var. Trans Size	low	low
R2	Sender Penalty for Var. Jitter	low	low
R5	Arbitration Fairness for GL Senders	low	high
R5	Arbitration Fairness for GT Senders	medium	high
R4	Performance Penalty for others senders in setups with jitter and var. trans	no	no

Table 3.1: Evaluation of the baseline NoC architecture features with and without the control layer (part 1).

Relevant Requirement	Feature \Mechanism Type	Baseline (PS + VC + RL, static priorities)	RM + Baseline NoC (PS + VC + RL, static priorities)
R7	Possibility to switch-off QoS	yes (but BE workloads may suffer from the high performance penalty)	yes
R7	Hardware overhead in setups with disabled QoS	medium (but workloads may suffer from the high jitter and out-of-order arrival of packets)	low (most resources from the control layer can be re-assigned for other purposes, opportunity to introduce a round-robin based scheduling on top pf routers with SPP)
R8	Support for Scalability	poor (but statically limited to the available HW resources)	good (largely independent of the underlying HW resources e.g. priority-based sharing of the same VC)
R8	Scalability Temporal Penalty	low (if there is enough HW resources)	medium (depends on the protocol overhead number and frequency of synchronized reconfigurations)
R8	Scalability Resources Penalty	high (a VC per priority level)	low (proportional only to the load and independent of resources)
R9	Support for Locality	no (yes only for highest prio)	yes (RM decides about the order of the transmissions therefore non-preemptive priority based schedules are also possible)
R10	Pessimism of formal Verification Static Setups	medium (depends on the HW resources in NoC, num. of VCs and size of buffers, low if there are enough resources otherwise high)	low (application of the global scheduling allows removing cyclic dependencies, efficiently capture dynamics)
R10	Complex Formal Verification Static Setups	low (complexity depends on the amount of available NoC resources low if there are enough resources high otherwise)	medium (complexity depends directly on the complexity of the introduced synchronization protocol)

Table 3.2: Evaluation of the baseline NoC architecture features with and without the control layer (part 2).

Chapter 4: Realization of the Control Layer in NoC

The control layer employs a formal, model-based verification for proving the adherence of the interconnect to real-time and/or safety constraints whenever a dynamic adaptation of resource allocation is necessary. From the conceptual point of view, the mechanism is based on a mathematical model providing the quality-of-service (QoS) abstraction of the underlying NoC (i.e., data plane). The workflow for a successful deployment of the proposed scheme is shown in Figure 4.1 and can be divided into the following four steps:

Step 1: Evaluation of the underlying MPSoC concerning timing properties.

Step 2: Development of the RM protocol/architecture for the selected MPSoC.

Step 3: Evaluation of the introduced protocol concerning temporal overhead.

Step 4: Assessment of the necessary HW extensions regarding implementation overhead.

In the first step (**Step 1**), it is necessary to verify to what extent the underlying MPSoC architecture is already suitable for hosting of the real-time and/or safety-critical workloads. In this context, the predictability of a NoC denotes the accuracy of the formal evaluation (i.e. calculation or "prediction") of the observed run-time behavior. The accuracy is evaluated for selected NoC properties (e.g. latency of transmissions) without (full) knowledge of run-time workloads (e.g. deployed traffic, senders behavior), cf.[45]. This includes assessment of the following NoC features:

- (flow-control) arbiters in routers,
- mechanisms for admission control in NI.

Note, that the predictability of a NoC does not guarantee a successful deployment of the control layer but provides important information about how difficult it is to compute the timing bounds for the NoC and how tight these guarantees are. Tightness refers to over-estimations which may happen during the analysis. For instance, the designer (in real-time domains) frequently considers only one potentially unrealistic scenario which can be worse than any observed combination of traffic at run-time, cf.[45]. The main trade-off lies between the quality of analysis results and the complexity of the computation. Usually, by increasing the complexity of the analysis one may improve its results, e.g., provide models which mimic the behavior of the NoC and traffic with higher accuracy. These inputs allow verifying if support for the control layer is possible in the context of a particular NoC and what benefits it could bring.

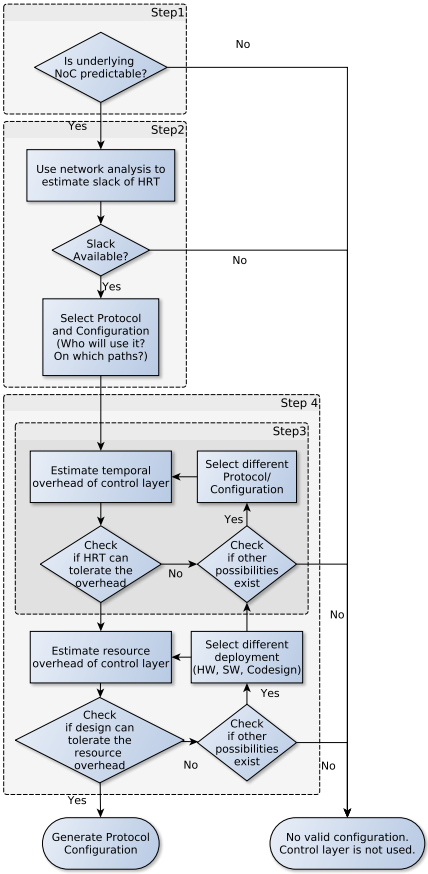


Figure 4.1: Design steps necessary for the implementation of the control layer.

To achieve the goals mentioned above, the initial evaluation employs profiles of senders (i.e., benchmarks) and models of the underlying NoCs' hardware (i.e., routers and network interfaces). The profiles of senders (e.g., behavioral models) can usually be extracted from the specifications of real-time and safety-critical applications. Recall, that the behavior and characteristics of such senders are typically well described and tested due to the nature of controlled processes and certification purposes (Chapter 1.4).

Examples for the characteristics are the maximal and minimal frequencies with which traffic is initiated, the number of initiated transmissions and the upper limits (i.e. deadlines) for the allowed transmission latencies. The non safety-critical,

best-effort (BE) senders constitute a special group of applications which can be integrated within a NoC-based system. They typically do not provide safe application profiles, e.g., general purpose applications running on processors with caches. For simulation of these applications, the synthetic benchmarks can be used (e.g., bursty access patterns using Markov chains) or traces of memory accesses recorded in real systems.

The mapping of applications to the network nodes plays an important role in the process of NoC design. In many existing NoC-based MPSoCs, nodes are heterogeneous and constructed out of processing cores as well as peripherals, e.g., I/O interfaces, Ethernet or DRAM controllers. These different hardware units have a fixed position in the system. Therefore, when applied to real-time setups, selected paths for critical communication are also fixed and enforced by the routing algorithm, e.g., transmissions from an Ethernet controller to the DRAM memory. Consequently, mapping or routing modifications cannot solve many of the performance bottlenecks which may occur in NoC.

The results from *Step 1* provide the designer with following outcomes:

- real-time models of the underlying NoC architecture (QoS parameters in routers and NIs which can be controlled with RM and clients) and sender profiles (e.g., worst-case latencies, deadlines, slacks, etc.)
- synchronization scenarios, i.e., sets of interfering senders (cf. Chapter 3)
- Quality-of-Service settings for each synchronization scenario assuring a predictable system behavior, e.g., rates enforced by NIs

In **Step 2** the design of an appropriate synchronization protocol and control layer elements, i.e., clients and RM, will be done. In this step, it is necessary to identify parts of the NoC architecture which can/must be controlled by the RM, e.g., rate limiters, admission control or settings of schedulers in routers. Note, that the RM can usually control a subset of these parameters without additional hardware overhead, i.e., there is no need for new interfaces or APIs. For instance, in many architectures, parameters for rate limiters are already propagated using control messages during the system startup. However, it may happen that some QoS parameters, although theoretically adjustable, would require slight modifications of the existing infrastructure, e.g., extending a router with a configurable flow table similarly to the Software-defined Networking (SDN). With this knowledge (about the elements that can be adjusted) it is possible to design the RM protocol, following the arbitration guidelines from Chapter 3. The protocol description should include the workflow, definitions of control messages, and arbitration in RM. The

average performance can be evaluated in a simulator (cf. Section 4.1.2) and the worst-case verification with a formal analysis framework (cf. Section 4.1.1).

In **Step 3** the derived protocol architecture will be evaluated concerning the introduced overhead. The QoS control plane, like many other mechanisms for QoS, introduces additional latency resulting from the synchronization. This overhead must be compared to the gain in guaranteed performance achieved using a RM-based arbitration. The results from related research (cf. Chapters 2 & 3) have shown that a RM-based control offers not only higher NoC's performance/utilization but, even more importantly, shorter, formally guaranteed latencies for realistic levels of utilization. Therefore, the reduced transmission latency can typically compensate for the overhead. Consequently, the utilization of the control layer enables solutions for otherwise unfeasible configurations. Additionally, Step 3 introduces an iterative process in which throughout gradual improvements a compromise between protocol efficiency and complexity (i.e., introduced overhead) can be found. The outcome of this design stage provides detailed requirements for the implementation of the platform extensions required by the control layer, e.g.: the number of synchronization scenarios, the frequency of initiated messages for estimating the size of the queues in the RM and clients, the maximum latency for message processing done by the RM and clients, and the maximal mode change latency (if applicable). Here, once again the introduced design tools can be used, i.e., worst-case analysis and simulation (cf. Section 4.1).

Finally, in **Step 4** the actual implementation of the clients and the RM is done, using the outcomes of the previously described design stages. The resource overhead resulting from the integration of the QoS control plane must be considered in the context of a concrete MPSoC. For instance, clients and RM modules can be realized in software (e.g., stand-alone libraries, extensions of the existing operating system (OS)/Run-Time Environment (RTE)) as well as in hardware (e.g. standalone modules, extensions of NIs). The former offers full flexibility, the latter maximum performance. Note that hybrid solutions (hardware/software co-design) re-using existing chip components are also possible. For example, as the complexity of a central RM unit is higher than the complexity of a client, the implementation can follow in the form of a stand-alone processing node, e.g., by re-using supervisor/-manager nodes available in many MPSoCs. The generation of HW/SW modules for the control layer (RM and clients) can be automatized with Electronic Design Automation (EDA) tools (or their extensions) for the configuration and design of the protocol. The discussion of the implementation following hardware/software co-design principles is presented in 4.2.2 and HW implementation follows in 4.2.3. The implementations of the control layer are flexible and may be carried out with different degrees of complexity. This permits a fine-granular control of the nec-

essary system resources. Consequently, Step 4 introduces iterations during which different possible setups must be evaluated to find a compromise between efficiency and overhead. In some cases, regressions to Step 3 are necessary as the introduced temporal overhead strongly depends on the method of implementation.

4.1 Design Environment

The design of a multiprocessor system-on-chip is a complicated process as on-chip networks integrate a plethora of programmable ECUs as well as other intellectual property (IP) components and peripherals. For evaluation of possible setups (e.g. NoC based MPSoCs running specific benchmark) there are mainly two tools available: *simulation* and *analysis*. The former is the state-of-the-art solution frequently used for verification of MPSoC's performance, e.g., Mentor Graphics Seamless-CVE , Axys Design Automation's MaxSim. Existing tools support cycle-accurate simulation of complete hardware and software of the system. Although simulation is time-consuming, it evaluates simultaneously (the same environment and benchmarks) the functioning and performance of the target. However, the effectiveness of this approach depletes along with increasing complexity of the simulated system. For a high number of stimuli (e.g., different arbiters, complex traffic patterns, etc.) it is often difficult to assess if the observed/measured results covered all possible corner-cases, i.e., the actual worst-case scenario. Therefore, the existing tools for simulation usually offer only quick and rough average system estimates but do not reliably cover system-level corner cases.

In contrast, a formal analysis, as the example described in [142], offers the full corner-case coverage and the worst-case performance bounds for the critical performance parameters. However, this technique also has a disadvantage. The provided guarantees are frequently pessimistic, i.e., they may cover scenarios which will never occur at runtime. This can lead to significant differences between the formally analyzed worst-case timing and the observed worst-case results in simulation or measurement. For instance, as reported by Daimler in [118] the majority of OEMs accept that the cyclic load of a bus should not exceed 50%. These differences are even higher in case of interconnects dominated by event-driven traffic, i.e., the theoretical worst case load may overshoot the actual utilization by 100% and even reach the level of 500%, cf. results from Daimler [118], similar Bosch [156].

In order to avoid resource over-provisioning and still be able to provide worst-case guarantees both tools are frequently used in conjunction. Consequently, the next sections describe the event-based simulator using the Omnet++ framework (cf. Section 4.1.2) as well as the formal analysis framework based on Compositional

Performance Analysis (CPA) (cf. Section 4.1.1) designed for evaluation of the control layer proposed in previous chapters.

4.1.1 Temporal Analysis

From a conceptual point of view, the introduced NoC control employs a formal, model-based verification to prove the adherence of the interconnect to the real-time and/or safety constraints whenever a safe adaptation is necessary. The mechanism is based on a QoS abstraction of the underlying NoC (data plane) allowing a path oriented arbitration approach. To assure safety, the calculation of the settings is done offline (at the design time) and is largely based on system-level, worst-case timing analysis methods. The analysis uses profiles of senders and models of the underlying NoC's hardware. The modeling of the underlying arbiters in routers can be conducted with any of the available frameworks e.g. [44],[89], [130].

The profiles of real-time and safety-critical senders can be typically extracted from the specification as the behavior and characteristics of real-time applications are usually well defined and tested. Examples of the characteristics are the maximal and minimal frequencies and the number of initiated transmissions (e.g., amount of transmitted data) as well as the upper limits (i.e., deadlines) for transmission latencies. These models can be later translated into the settings for the rate controllers on each processing node, i.e., the system must assign a sufficient bandwidth share to reach its deadlines and requested throughput for each real-time and/or safety critical sender.

Using these inputs, the worst-case analysis derives information about the schedulability of integrated workloads, e.g., if all real-time transmissions meet their deadlines. This is done through maximization of the response time. The response time denotes the period between the beginning (activation) and termination of each transmission. Additionally, the analysis provides indications about the resources required for achieving these results, e.g., the maximal number of outstanding packets in routers buffers.

These initial results are used as a foundation for the incorporation of the control layer in the design. The online adaption of the system behavior with the control layer will induce additional overhead to the system. In this context, it is possible to distinguish between two major sources of overhead resulting from the control layer: (i) control messages (i.e., transmission latency, interference), and (ii) complexity of used protocol (e.g., the different system behavior due to the protocol, frequency of mode changes). These factors must be accounted for evaluation of the systems worst-case performance. Safe application of the control layer is possible whenever all real-time senders belonging to the synchronization scenario

have enough slack to compensate protocol overhead. The slack denotes the difference between the maximum time needed to traverse the network and the allowed deadline for traversing the network. The formal NoC analysis of the underlying architecture is used as an input to the analysis of the RM arbiter and the control layer. The analysis models/frameworks for different synchronization protocols are provided in the related work, e.g., round-robin [81], static priorities [83], dynamic priorities [75]. As these are based on well-known principles that are also used by commercial tools (e.g. SymTA/S, Timing-Architect Tools Suite), it is possible to integrate/provide extensions for formal verification of the control layer in existing, commercial toolchains/workflows.

Note, that the worst-case latencies of transmissions synchronized with RMs depend directly on the activities of other senders sharing the resources. Therefore, it is possible to apply temporal-analysis frameworks, such as Real-Time Calculus [12] or Compositional Performance Analysis (CPA) [61], which use abstractions of the worst-case possible access traces (event models), to capture the dynamics of the system behavior. This reduces the pessimism in setups which do not fully utilize the resources. The main goals of the formal analysis are:

- evaluation of the profit resulting from the control layer with respect to the worst-case performance of senders
- evaluation of the temporal overhead resulting from the transmissions of the control messages and the selected arbitration scheme

Through iterative runs and design space exploration the designer should search for the configuration and synchronization method which would achieve schedulability of the system and improve its performance. As a result of the worst-case analysis the designer obtains the following information:

- worst-case performance of the NoC with the applied control-layer,
- settings for the synchronization protocol and senders which meet the temporal requirements of the workload.

Recall, that providing the formal guarantees is the predominant requirement for safety-critical domains. The effectiveness of the simulation for evaluation of the worst-case behavior of complex setups is limited. The main problem results from the necessity of providing the stimuli to cover all system-level corner cases.

The following sub-sections are structured as follows: Subsection 4.1.1 introduces the basic principles of the Compositional Performance Analysis framework. This

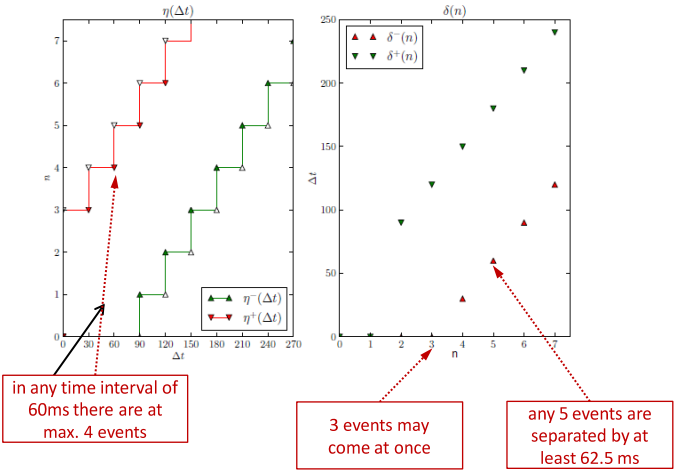


Figure 4.2: Event models covering all traces which stay between $\eta^+(\Delta t)$ / $\delta^-(n)$ and $\eta^-(\Delta t)$ / $\delta^+(n)$.

analysis framework along with the system model presented in Section 4.1.1 constitute the main input for the analysis of the introduced control layer from Section 4.1.1. The evaluation of different scheduling policies for the operation of control layer is presented in Chapter 5.

Compositional Performance Analysis

The proposed analysis is based on the Compositional Performance Analysis (CPA) framework [61] and is designed to evaluate the worst-case performance of the control layer. The description in this section is based on [137] and [61]. The system model introduced by CPA is built out of the three main components: resources, tasks, and event models. Resources are used for modeling of processing or network nodes (e.g. CPUs, router ports, control units). Tasks are mapped to resources and compete for the service provided by them. The allocation of the service depends on the selected scheduling policy.

To capture the dynamics of the system's behavior, tasks activations are abstracted using event models. These models define arrival functions $\eta^-(\Delta t)$ and $\eta^+(\Delta t)$, which provide the lower and upper bounds on the number of events (task activations) in any half-open time interval. Consequently, they allow to capture all possible event arrival patterns/scenarios within interval bounds. Therefore, it is

possible to cover all corner-cases, and not only a single scenario as in the case of a recorded memory trace i.e. model variety of activation patterns used in practice e.g. periodic + spontaneous, dual cyclic, TDMA. Correspondingly, the minimum and maximum distance functions $\delta^-(n)$ and $\delta^+(n)$ are counterparts of event arrival functions η defining the lower and upper bound on the time interval between the first and the last event in any sequence of n occurrences of events. As presented on Figure 4.2 both η and δ functions can be straightforwardly converted to each other. The summary of methods for obtaining typically used behavioral models is available in [121], [33], [128].

The execution of an application in a system is modeled using a directed graph. In such setup, edges symbolize dependencies and nodes denote tasks. A task consumes temporal service of a resource which varies per activation between best- and worst-case execution/transmission time. The jitter (the difference between maximum and minimum response times) resulting from the response time permits deriving new output event models. Consequently, an output event model of a task on a particular resource becomes the input event model for its dependent task(s) on another resources. Additionally, the CPA framework efficiently analyzes the functional dependencies between tasks such as communication/execution chains e.g. activations of tasks which depend on inputs from multiple other tasks. This is carried out by joining the event models from several tasks using AND function [61]. CPA offers an iterative approach following the busy period method [139]. According to this approach, the critical instance scenario is used for derivation of the formal guarantees. It maximizes the response time of the currently analyzed task by simultaneous activations of all other interfering tasks and assuming a maximal load initiated by them (i.e. worst-case activation patterns).

CPA propagates event models between dependent tasks in a loop. This is done in form of an iterative process. The analysis finishes when all models reach a fixed-point and do not change anymore. Alternatively, the analysis could finish earlier when a constrain violation is detected e.g. too high number of analysis iterations or end-to-end latencies, or too high response time.

Modeling Control Layer with Compositional Performance Analysis

Our starting point is a system with a set of real-time tasks/applications $T = \{\tau_1, .., \tau_l\}$ mapped on a multicore platform composed of n processing nodes $P = \{p_1, .., p_n\}$. These cores are connected through a Network-On-Chip (NoC) composed of k routers $R = \{r_1, .., r_k\}$. As detailed in Chapter 3, this work concentrates on the NoCs with work-conserving schedulers i.e. performance oriented router arbiters or prioritization applied for independent VCs (cf. Section 3.1).

Definition 1 (Performance Optimized Networks-on-Chip) *The considered class of NoCs is modeled as a tuple $N = (P, R, V, L)$ where P is a finite set of processing nodes, R is a finite set of routers, V is a set of virtual channels and $L \in R \times R$ is a set of communication links all with the same bandwidth.*

The model of a NoC with priorities assigned to the Virtual Channels can be straightforwardly derived from the model of the performance optimized NoC.

Definition 2 (Priority Based Networks-on-Chip) *The considered class of NoCs is modeled as a tuple $N = (P, R, V, L, \Theta)$ where P is a finite set of processing nodes, R is a finite set of routers, V is a set of virtual channels and $L \in R \times R$ is a set of communication links all with the same bandwidth.*

The set of priority values φ is partitioned into disjoint sets. The function $\Theta : V \rightarrow \varphi$ assigns to each virtual channel v_i a range of priority values $[\underline{\varphi_i}, \bar{\varphi}_i] \subset \varphi$. By assigning ranges of priority levels to VCs, the hardware can support more criticality levels than available VCs. Therefore a finite set of priorities is assigned, which corresponds to the available set of virtual channels. Note that router and the processor(s) connected to it through the network interface are referred similarly.

The traffic in the NoC is determined by a set of data streams between communicating tasks. The following assumes *arbitrarily* activated tasks/data streams. The dynamics of the communication traffic is modeled using event-arrival functions where an event refers to a transmission (composed of one or multiple packets) traveling through the network from source to destination.

Definition 3 (Data Streams) *A data stream S defined between a couple of tasks (τ_1, τ_2)[1] is a tuple $S = ((T \times T), \varphi, n, \eta(\Delta t), D)$ where T is a finite set of tasks including τ_1 and τ_2, φ is a statically assigned priority level equal to the priority level of a sending task, $n \in \mathbb{N}$ is a constant which defines the amount of communicated data in number of packets and $\eta(\Delta t)$ is an event-arrival function which defines a bound on the number of events which can arrive in a given time interval Δt. Each data stream has a real-time constraint, i.e a deadline D over the latency of the transmission through the NoC.*

Note that all packets/flits of a given stream inherit the priority level of this stream. Task mapping and routing strategy determine the link between data streams and the hardware NoC mapped on it.

Definition 4 (Transmission) *A transmission m of a data stream S on a network-on-chip N is defined by the mapping function $m : S \longrightarrow N$ which assigns to every couple*

[1] τ_1 sends data to τ_2.

of tasks $(\tau_i, \tau_i') \in e_1(S)$[2] *the network nodes* $(s, d) \in e_1(N) \times e_1(N)$ *on which they are respectively executed, a path* $h = \{(s, r_1), (r_2, r_3), ..., (r_k, d)\}$ *in the network and a virtual channel* $v_k \in e_2(N)$ *according to the appropriate priority value (if supported by the NoC) such that* $\varphi \in \Theta(v_k)$. *Note that the assigned virtual channel is the same for all routers* r_j *in the path* h.

Additionally, the priority level of several transmissions can fall into the same range of values of a given VC and therefore corresponding streams will be assigned the *same* VC. If these streams share the same path, the control layer must be used to arbitrate between them and to provide timing guarantees to each stream. Therefore streams are clustered in *synchronization scenarios* which require synchronization with RM. Moreover, in some designs multiple VCs may have the same priority. In this case, the fair treatment is assumed using a selected local arbitration method in the router.

Definition 5 (Synchronization Scenarios) *A synchronization scenario* $r = \{m_1, m_2, ..., m_n\}$ *is a set of* n *transmissions sharing the same VC and at least one link in their path. That is,*

$$r = \bigcup_{i,j \in n \times n} (m_i, m_j) \quad s.t \quad v_i = v_j \text{ and } (h_i \cap h_j) \neq \emptyset$$

where $m_i = ((s_i, d_i), v_i, h_i)$ *and* $m_j = ((s_j, d_j), v_j, h_j)$. *Note that a synchonization scenario is associated to one VC.*

There is a finite number of such scenarios in the system i.e. in the worst-case scenario all transmissions for one synchronization scenario. As mentioned previously, this work assumes that the baseline architecture of a particular NoC is analyzable. The analysis provides *basic* worst-case latency bounds for any transmission conducted in the network. The bottom-layer latency is determined by the following factors: the source/destination routing distance, packet size, link bandwidth, additional protocol overheads and other ongoing communication.

Definition 6 (Basic Network Latency Bounds) *Let the minimum and maximum network latency* C_i^- *and* C_i^+ *of a given transmission* m_i *denote the minimum and maximum time required to transfer* n *packets from source to destination.* C_i^- *corresponds to the scenario where there is no contention in the network and* C_i^+ *to the scenario where contentions from other VCs are involved.*

Note that worst case latency bounds involving contentions from the same VC are provided by the timing analysis of the control layer.

[2]The notation $e_i(S)$ refers to the i-th item of a tuple S.

Corollary 1 *The time during which the packets from the transmission m_i are physically present in the network can be bounded by $[C_i^-; C_i^+]$.*

The selected analysis of the baseline NoC architecture must be capable of providing basic network latency bounds. Obviously the worst-case guarantees depend on the quality of this analysis. This work relies on the analysis from [142] and presents in the rest of the chapter how to incorporate C_i^+ and C_i^- in the analysis of the control layer. Finally, each transmission with real-time requirements must finish before a given deadline D_i.

Analysis and Derived Metrics

As discussed, the proposed mechanism can be efficiently analyzed using the busy window approach (see [91],[61]). It constructs a critical instant (i.e. the beginning of the busy window) and considers the worst-case activation sequence of all transmissions, their duration, and the scheduling policy to compute the maximum delay a sender can experience. Therefore, this method maximizes the response time, denoting the duration between the beginning (activation) and termination of a transmission. Every path used by senders with real-time and/or safety requirements is considered to be a *resource* under a selected arbitration method (i.e. scheduler) and every interfering sender is considered to be a *task*. In the following of this sub-section, the analysis introduces basic definitions (e.g. busy-window, response time) for transmissions in a system *without* the control layer. Later, they will be extended to account for blocking from the specific synchronization method and protocol overhead.

Definition 7 *Let $\omega_i^-(q)$ and $\omega_i^+(q)$ denote the minimum and maximum q-event busy window of a sender i on a path h, which is the time interval required to fully transmit q consecutive packets from the sender on that path.*

Theorem 1 *The worst-case time necessary to transmit q consecutive packets on path h considering interfering senders on the same path is bounded by [77]:*

$$\omega_i^+(q) \leq q \cdot C_i^+ + B_i(w_i^+(q)) \tag{4.1}$$

where $B(w_i^+(q))$ is the maximum interference resulting from other traffic issued by other senders from the synchronization scenario sharing the same path.

Proof: The busy window of q consecutive packets from a sender i on a path h is bounded by the time necessary to send all packets ($q \cdot C_i^+$) on the selected path and the maximum blocking time due to interfering traffic on the shared path ($B_i(w_i^+(q))$). ■

As the $\omega_i^+(q)$ appears on both sides of Eq. 4.1 this equation represents an integer-fixed point problem. It can be solved iteratively starting with $\omega_i^+(q) = q \cdot C_i^+$. The interference factor $B_i(w_i^+(q))$ depends on the selected arbitration method (scheduler). In depth analysis of different arbitration methods will be provided in the next sub-section. Note, that control layer does not cancel the interference from synchronized senders ($B_i(w_i^+(q))$) but moves it from the NoC to the cores to avoid propagation of blocking.

Based on the computed busy-window $w_i^+(q)$, it is possible to derive the *the worst-case response time* R_i^+ of a single packet and of each activation q in the busy-window $R_i(q)$ from sender i, i.e., the longest time interval between activation and completion of a transmission on a given path:

$$R_i^+ = \max_{\forall q \geq 1} \left\{ R_i(q) \mid \omega_i^+(q) \geq \delta_i^-(q+1) \right\}$$
$$R_i(q) = \omega_i^+(q) - \delta_i^-(q) \tag{4.2}$$

The response time R_i^+ is represented by the difference between the busy-window $\omega_i^+(q)$ and the earliest possible activation $\delta_i^-(q)$ of a packet. This difference bounds the maximum time $l_i^+(q)$ necessary for a sender i to fully transmit q packets over the selected path (cf. [142]) by:

$$l_i^+(q) \leq Inj_i^-(q) + R_i^+, \tag{4.3}$$

where $Inj_i^-(q)$ denotes the time the sender intends to inject the packets. Basically the equation computes the time interval required by a stream to inject q packets and then assumes the last one of these to experience the worst-case blocking in the network. Due to the in-order delivery of the network, all previous packets must have arrived at the destination before the last one. The additional delay which previous flits may observe is included as interference in the worst-case blocking of the last flit.

Based on the worst-case time to transmit q consecutive packets $\omega_i^+(q)$, follows the lemma of the minimum accepted throughput $\hat{\beta}^-$ for a sender i:

$$\hat{\beta}_i^-(\Delta t) = \min \left\{ \Delta t - 1, \max \left\{ n | \omega_i^+(n) < \Delta t \right\} \right\} \tag{4.4}$$

This lemma can be proven as follows: The *max*-function selects the highest number of events that can be accepted (i.e. fully transmitted) during a time interval Δt based on the busy-window of sender i. In order for this to happen, only events that can arrive before Δt can be accepted. Additionally, the network cannot accept more packets than time slots have passed, covered by the first term of the *min*-function.

These calculations allow defining different network slacks for a (critical) sender i on a given path. The worst-case network slack of a single packet $\hat{S}_i^-$, of packet q in a transmission $S_i^-(q)$, and of a transmission considering q packets:

$$\begin{aligned} \hat{S}_i &= D_i(1) - R_i^+ \\ S_i(q) &= D_i(q) - R_i(q) \\ \bar{S}_i(q) &= \bar{D}_i(q) - l_p^+(q), \end{aligned} \tag{4.5}$$

where $D_i(q)$ is the deadline of the q-th packet in a transmission (derived from the real-time requirements), $D_i(1)$ the deadline for a single packet, and $\bar{D}_i(q)$ the end-to-end deadline for the transmission of q packets. The slack is basically the difference between the maximum time needed to traverse the network and the allowed deadline for traversing the network.

The slack and minimum accepted throughput are computed offline and can be used during the system design phase for the worst-case dimensioning. Depending on the design goal, one of the following requirements or any combination must be satisfied for each critical sender for a system to be schedulable:

$$\begin{aligned} &(r1) \quad D_i(1) \geq R_i^+, \\ &(r2) \quad D_i(q) \geq R_i(q), \\ &(r3) \quad \bar{D}_i(q) \geq l_i^+(q) \\ &(r4) \quad \forall t \in \mathbb{N}^+ : \ \hat{\beta}_i^-(t) \geq \alpha_i(t-d) \end{aligned}$$

where α_i denotes the requested throughput of sender i and d is a specified delay for the sender until which the network needs to provide the required service [142].

If all or some HRT senders have an available slack, this can be used for the control layer presented in Chapter 3. Based on these conditions and specifications, the analysis can be applied during the design phase of the system to check if the control layer can be safely deployed and which protocol and configuration are allowed. If the temporal overhead of the control layer is acceptable, it can be safely used to dynamically adapt the system behavior and increase the performance of the system under presence of the dynamics.

Blocking and Timing Overhead of the Control Layer

The online adaption of the system behavior and the control layer will induce additional overhead to the system. Regarding the timing overhead, it is possible to distinguish between two major sources of overhead resulting from the control layer: (i) control messages (i.e. transmission latency, interference), and (ii) complexity of used protocol (i.e. the different system behavior due to the protocol, frequency of mode changes etc.). These must be accounted for when analyzing and

designing a system with the proposed resource arbitration scheme. Note that, depending on the implementation, the aforementioned sources can have a different impact on the system.

The overhead of the control messages depends on the implementation of the control layer, which is discussed in detail in the next section. For instance, if a dedicated network/communication media for such messages exists, they will have no direct impact on the timing (i.e. they induce no additional interference in the network). However, if a network with virtual channels is used and the control messages are sent on a higher priority level than the data, they induce an additional interference. This must then be accounted for, when analyzing the latency and the possible interference with other tasks. The latter is important, as the additional interference from the control messages will increase the time a sender spends in the network (i.e. the busy-window). And thus, it might observe a higher number of interfering packets from another sender.

The protocol overhead depends on the actions necessary for conducting mode-changes with respect to their frequency and complexity. For example, if the protocol delays the access to the network to ensure a path can be used in isolation by a sender, there will be an initial delay before accessing the network. However, as this delay is spent at the sender, it will not increase the overhead a sender will experience in the network. Yet, if the protocol which permits the sharing of some links will be applied (instead of detouring or blocking other traffic), this time will count as an additional interference increasing the time the transmission will spend in the network. And thus it might also increase the number of packets from another sender that can be observed while being in the network. To estimate the overhead of the different approaches, the analysis will be extended to account for the control layer.

Definition 8 *The Protocol Overhead O_i denotes the additional delay which a single transmission (possibly containing multiple packets) from an application i may experience in the worst-case due to concurrently running interfering senders on the selected path.*

Consequently, the proposed mechanism allows safe application of the control layer whenever all critical senders belonging to the synchronization scenario have enough slack to compensate protocol overhead, i.e., satisfy the requirements when accounting for the overhead:

$$
\begin{aligned}
&(r1) \quad D_i(1) \geq R_i^+ + O_i, \\
&(r2) \quad D_i(q) \geq R_i(q), \\
&(r3) \quad \bar{D}_i(q) \geq l_i^+(q) + \left\lceil \frac{q}{n_i} \right\rceil \cdot O_i,
\end{aligned}
$$

where n_i is the number of packets in a transmission of stream i and hence $\lceil \frac{q}{n_i} \rceil$ the number of transmissions needed for q packets. Note that depending on the implementation of the control layer, the protocol overhead might contain the control messages of other senders i.e. preemption time due to activations of senders with higher priority. Moreover, the needed requirements or a combination of requirements strongly depend on the system design and thus not all requirements need to be satisfied in all cases.

Analysis for Different Allocation Schemes

This sub-section presents an analysis of different resource allocation schemes for variants of mechanism presented in Chapter 3. The application of the mechanism to real-time systems requires a formal analysis capable of providing latency and throughput guarantees for the synchronized transmissions[3]. For analysis purposes, every synchronization scenario protected by a RM is considered to be a resource under scheduling and every transmission belonging to this scenario as a task. The *release time* of a task is represented by the arrival of the request message (reqMsg) to the RM. The time which a request has to wait to use a resource (scheduling and preemption delay) is denoted as *blocking time* of the task on the resource. Task *execution* constitutes the time necessary for sending all packets belonging to a particular transmission and releasing the resource. The formal worst-case timing analysis must account for latencies resulting from the synchronization protocol which are defined by the architecture of the interconnect.

In case of the RM-based NoC implementation this is not fully true due to protocol overvead.

Theorem 2 *The worst-case time necessary to conduct q transmissions i belonging to a synchronization scenario protected by the resource manager RM is bounded by:*

$$\omega_i^+(q) \leq q \cdot C_i^+ + O_i(\omega_i^+(q)) + B_i(\omega_i^+(q)) \tag{4.6}$$

where, C_i^+ denotes the maximal transmission latency, O_i denotes maximum protocol overhead for sender i and $B_i(\omega_i^+(q))$ the maximum blocking resulting from scheduling of other transmissions belonging to the same synchronization scenario.

Proof: The theorem directly results from the description of the mechanism and protocol (see Sec. 3.6.1). The busy window of q consecutive transmissions m_i is

[3] For interested reader, the detailed discussion of rea-time metrics applicable to NoC traffic is presented in Chapter 2

bounded by the time necessary to send all packets belonging to these transmissions ($q \cdot C_i^+$), the time (O_i) necessary to exchange control messages for each transmission, plus the maximum time interval during which a particular request can be blocked due to other ongoing transmissions. ■

Note that the $\omega_i^+(q)$ appears on both sides of Eq. 4.6 forming a fixed point iteration problem. It can be solved iteratively starting with $\omega_i^+(q) \leq q \cdot C_i^+$ and passing the result to the $O_i^+(\omega_i^+(q))$ and $B_i(w_i^+(q))$ functions. Later the calculations must be repeated until two consecutive iterations produce the same result. Additionally, the number of activations q during the time interval Δt can be calculated using $\eta^+(\Delta t)$, therefore both notations can be used interchangeably.

In the following, the in-depth discussion of B_i and O_i factors for different synchronization methods will be provided.

Time-Division Multiple Access

The description begins with time driven-scheduling which is based on two following standard notions of time allocation: time slot and scheduling cycle. The following definitions will be used later in this work.

Definition 9 (Time slot) *A time slot s is the basic time budget allocated to each application belonging to the synchronization scenario S, during which it acquires an exclusive access to the NoC. It is assumed that this static parameter is assigned at design time and does not change during execution. Note that each transmission can have a different size of the time slot. This can be used to distinguish between data streams and favor one application, over other applications accessing the NoC.*

The time slot can be obtained from the analysis of the underlying NoC architecture (i.e. basic network latency).

Definition 10 (Scheduling Cycle) *The time to access the NoC is therefore partitioned into a set of time slots allocated to applications in a synchronization scenario as follows,* $\Theta : \{s_1, s_2, ..., s_k\} \longrightarrow \{t_1, t_2, ..., t_n\}$ *where* $\Theta(s_i) = t_j$ *means that a time slot* s_i *is allocated to transmission* t_j*. Note that the same time slot cannot be allocated to different applications. A scheduling cycle defines a cyclic, repetitive order according to which the same application is allocated access to the NoC. That is,* $\Theta(s_i + \delta) = \Theta(s_i)$*,* δ *being the length of the cycle.*

In a standard time driven allocation (e.g. round-robin or TDM), the duration of the scheduling cycle is static and equal to the number of synchronized applications. In the following pages, the terms **TDM-cycle** and **scheduling cycle** are used interchangeably.

The basic property of the TDMA is to enforce a fully static NoC behavior i.e. make transmission time independent of behavior of other running senders. Therefore, time slots are assigned to tasks regardless if the tasks are active, i. e. request the resource, or not. The blocking time, under the TDM-based scheduling, depends statically on the activations of other synchronized transmissions i.e. the length of the TDM-cycle does not change in time.

Lemma 1 *The interference which $q = \eta^+(\Delta t)$ requests experience in a time window Δt can be bounded by:*

$$B_i^{TDM}(\Delta t) = \eta_i^+(\Delta t) \cdot \left\lceil \frac{C_i^+}{s_i} \right\rceil \cdot \sum_{\forall j \in S} (s_j) \tag{4.7}$$

where S is a set of all transmissions belonging to the same synchronization scenario as transmission i, C_j^+ denotes the maximum latency of the transmission j , (s_j) denotes the maximal blocking caused by a single TDM-slot belonging to the transmission j, $\eta_i^+(\Delta t)$ denote the maximum number of initiated transmissions by the sender under analysis (i) within interval Δt.

Proof: The theorem results directly from the description of the scheduler (see Sec. 3.6.1). The sum computes the interference from all other transmissions belonging to the same synchronization scenario S. Following the TDM scheduling policy, each TDM slot s_j belonging to the transmission j may block only once a single slot belonging to the transmission i. Moreover, transmission j cannot block more slots than than the number of TDM cycles which it requires to completion i.e. $\eta_i^+(\Delta t) \cdot \lceil C_j^+ / s_i \rceil$). The assumed scheduler is a special case of the TDM-scheduler scheme which was analyzed in [121]. ■

Theorem 3 *The worst-case protocol overhead O_i^{TDM} in the TDM cycle during the period Δt can be calculated as follows:*

$$O_i^{TDM}(\Delta t) = \eta_i^+(\Delta t) \cdot \underbrace{\left(\left\lceil \frac{C_i^+}{s_i} \right\rceil (C_{i,ctrl}^+)\right)}_{\text{own activations}} + \underbrace{\left\lceil \frac{C_i^+}{s_i} \right\rceil \cdot \sum_{\forall j \in S} (C_{j,ctrl}^+))}_{\text{Interference from other senders}} \tag{4.8}$$

where $C_{i,ctrl}^+$ denotes the maximum latency of the control messages (gntMsg) and s_i the length of the TDM-slot for the considered transmission i. Note that different transmissions maybe assigned slots of different size.

Proof: The theorem results directly from the description of the mechanism and protocol (cf. Sec. 3.6.1). The protocol overhead of $q = \eta_i^+(\Delta t)$ consecutive activations of transmission i includes for each activation: the time necessary for

RM to issue *gntMsgs* for each time slot required for the particular transmission ($\left\lceil \frac{C_i^+}{s_i} \right\rceil \cdot C_{i,ctrl}^+$), and also control messages for granting time slots for slots of other synchronized senders $\left\lceil \frac{C_i^+}{s_i} \right\rceil \cdot \sum_{\forall j \in S} (C_{j,ctrl}^+)$). ■

Round-Robin Based Performance Optimized Arbitration

Round-robin schedulers (c.f. [92]) avoid a performance penalty introduced by the TDMA in dynamic setups (arrival jitter, variable transmission length) by releasing a time slot whenever they are not used by senders. Consequently, it reduces the unnecessary idle times and results in an improved, more compact schedule. However from the formal verification perspective, although round-robin systems perform much better than TDMA, on average, they still offer the same worst-case guarantees as TDMA schedulers i.e. the worst-case load and response time calculations are the same as in case of TDMA.

In case of the RM-based NoC implementation this is not fully true i.e. there is a difference resulting from a modified synchronization protocol - enhancement from one *gntMsg* control message to three: *reqMsg*, *ackMsg* and *relMsg*. The analysis is based on the protocol introduced in Section 3.6.1 and was published in [81] and extended in [82].

Lemma 2 *The blocking time which q requests experience in a time window Δt can be bounded by*

$$B_i^{RR}(\Delta t) = \eta_i^+(\Delta t) \cdot \sum_{\forall t_j \in S} (C_j^+) \cdot \min\{q, \eta_{j,out}^+(\Delta t)\} \tag{4.9}$$

where, S is a set of all transmissions belonging to the same synchronization scenario as transmission i, C_j^+ denotes the maximum latency of the transmission t_j and $\eta_{j,out}^+(\Delta t)$ is the maximum number of requests for transmission j within interval Δt taking into consideration the worst-case propagation jitter resulting from the transmission of control messages.

Proof: According to the assumptions (see 3.6.1), the considered RM uses a round-robin scheduler. The sum in Eq. 4.9 computes the interference from all other transmissions belonging to the same synchronization scenario. Following this arbitration policy, each transmission j may block only once the transmission i. However, $\eta_i^+(\Delta t)$ activations of i can be blocked only $\eta_i^+(\Delta t)$ times and also no more than $\eta_j^+(\Delta t)$. This is denoted by the min function. Finally, it is assumed that the first requests from all streams arrive exactly at the same moment in order to construct the critical instance. Therefore, it is a conservative overestimation of the actual interference. The assumed fixed round-robin scheduler is a special case of the round-robin scheme which was analyzed in [119] and also includes a proof. ■

Theorem 4 *The worst-case protocol overhead in the time interval Δt resulting from $\eta_i^+(\Delta t)$ transmissions i belonging to a synchronization scenario protected by the resource manager RM is bounded by:*

$$O_i^{RR} = \eta_i^+(\Delta t) \cdot (3 \cdot C_{i,ctrl}^+ + \sum_{\forall t_j \in S} (2 \cdot C_{j,ctrl}^+) \cdot \min\{\eta_i^+(\Delta t), \eta_j^+(\Delta t)) \qquad (4.10)$$

where, C_i^+ denotes the maximal transmission latency, $C_{i,ctrl}^+$ denotes maximum latency of the control messages for transmission i, $C_{j,ctrl}^+$ denotes maximum latency of the control messages for other transmissions belonging to the same synchronization scenario S.

Proof: The theorem directly results from the description of the mechanism and protocol (see Sec. 3.6.1). $\eta_i^+(\Delta t)$ consecutive transmissions must send $(3 \cdot C_{i,ctrl}^+)$ control messages for each transmission (reqMsg, ackMsg, relMsg), plus account for two control messages from each transmission from interfering senders i.e. *ackMsg* and *relMsg*. The *reqMsg* from interfering senders happens in parallel with ongoing transmissions and therefore does not influence the timing. The number of blocking is calculated as in Eq. 4.9. ■

In the following paragraphs, the work-conserving priority based schedulers will be described.

Mode Change Analysis

In contrast to time-driven schedulers, priority based schedulers introduce mode changes for the work of the system. For instance, whenever a RM is trying to block a certain sender it must send an appropriate control message. Until receiving this message, the client may freely use the NoC (inject packets) as it is unaware of the changed state of the NoC. Consequently, receiving the block message does neither guarantee nor prevent interference with packets injected in the previous mode of NoC work (i.e. when client was still allowed to use interconnect). Note, that in the majority of NoCs it is impossible to remove (once injected) flits from the NoC routers, one must wait until they leave the interconnect. This introduces additional delay: *mode-change latency*.

The RM must assure that the transition between modes is safe. Therefore, before switching on a new mode of NoC work (e.g. acknowledging senders with higher priority, change settings of rate controllers), the RM sends to the clients supervising active applications, a stop message (stopMsg) to block all accesses to the NoC from the corresponding node. Clients then wait for the confMsg communicating the current system mode. Consequently, the RM sends a confMsg with a delay (cf. Section 3.6.2,3.6.5) to ensure that packets from previously ongoing transmissions have left the NoC i.e. provide logical isolation. After receiving the confMsg, clients

adjust the rate and unblock transmissions. Additionally, the confMsg initiates active applications (the ones which just issued actMsg) which can then use the NoC.

Static Priority Preemptive Scheduler

This section introduces analysis of the static priority preemptive (SPP) arbitration from [83] done using the introduced mode-change analysis. The analysis extends the principles of commonly used SPP schedulers [92], [127], [106] by considering the overhead resulting from the introduced control layer and specifics of NoC architecture.

Firstly, the maximum blocking time which a transmission may experience due to other transmissions belonging to the same synchronization scenario will be considered. Unlike in the classic SPP scheduler, the preemptions introduce an additional delay also for applications with higher priorities. This preemption is caused by the synchronization protocol and the necessity to ensure that two transmissions will not interfere with each other i.e. prevent priority inversions.

Lemma 3 *B_i^{SPP} defines the worst-case blocking that can happen during time interval Δt in the system with RM applying static priority preemptive scheduler. It is build of two factors. Indirect preemption delay (IPD) defines the additional latency which $\eta_i^+(\Delta t)$ requests for a transmission i will experience in the worst-case due to transmissions with a lower priority. Direct blocking (DPD) which $\eta_i^+(\Delta t)$ requests experience in a time window Δt, due to transmissions with a higher priority. Consequently, the worst-case blocking can be bounded by:*

$$B_i^{SPP}(\Delta t) = \eta_i^+(\Delta t) \cdot \underbrace{(\min\{\eta_i^+(\Delta t), \sum_{\forall t_j \in lp(i)} \eta_j^+(\Delta t)\} \cdot (\max_{\forall t_j \in lp(i)} (C_j^+))}_{\text{Indirect Preemption Delay}} + \underbrace{\sum_{\forall t_j \in hp(i)} \eta_j^+(\Delta t) \cdot (C_j^+))}_{\text{Direct Preemption Delay}} \quad (4.11)$$

where, $lp(i)$ is a set of transmissions with lower priority than t_i belonging to the same synchronization scenario S, $hp(i)$ is a set of transmissions with higher priority than t_i belonging to the same synchronization scenario S, $C_{j,ctrl}^+$ denotes the maximum latency of the control message for transmission t_j, C_j^+ denotes the maximum latency of the single transmission t_j.

Proof: The proof is derived directly from the description of the mechanism. Each $\eta_i^+(\Delta t)$ transmissions t_i must preempt maximally $\eta_i^+(\Delta t)$ times other transmis-

sions belonging to $lp(i)$. It may also happen that, in the worst-case, the maximal number of activations of tasks from $lp(i)$ is lower than $\eta_i^+(\Delta t)$ therefore one should select the minimum from both. Secondly, the *RM* must assure the logical separation between preempted transmission and t_i i.e. that all packets from the preempted transmission t_j safely leave the NoC before t_i starts. This is conservatively assured by selecting the longest (i.e. maximum) transmission latency of the one (i.e. last) packet from all transmissions belonging to $lp(i)$.

In the next step follows the definition of blocking which a task may experience due to activations of other tasks with a higher priority belonging to the same synchronization scenario. Following the assumed static priority preemptive (SPP) arbitration policy, each transmission m_j may block/preempt several times the transmission t_i. However, the number of activations of t_j can be bounded by $\eta_j^+(\Delta t)$. ■

Next, the worst-case protocol overhead will be defined.

Theorem 5 *The worst-case protocol overhead in the time interval Δt resulting from $\eta_i^+(\Delta t)$ transmissions i belonging to a synchronization scenario protected by the resource manager RM in a system with static priority based scheduling is bounded by:*

$$O_i^{SPP}(\Delta t) = \eta_i^+(\Delta t) \cdot (\underbrace{3 \cdot C_{i,ctrl}^+}_{\text{own activations}} + \underbrace{\min\{\eta_i^+(\Delta t), \sum_{\forall t_j \in lp(i)} \eta_j^+(\Delta t)\} \cdot (\max_{\forall t_j \in lp(i)} (C_{j,ctrl}^+))}_{\text{Indirect blocking}} + \underbrace{\sum_{\forall t_j \in hp(i)} \eta_j^+(\Delta t) \cdot (C_{i,ctrl}^+ + 2 \cdot C_{j,ctrl}^+))}_{\text{Direct blocking}} \quad (4.12)$$

where, C_i^+ denotes the maximal network latency of m_i, $C_{i,ctrl}^+$ denotes maximum latency of the control message for m_i and $B_{i,IPD}(w(q))$ the maximum blocking resulting from preemptions of lower-priority tasks and $B_{i,DPD}(w(q))$ maximum blocking due to transmissions with a higher priority than m_i.

Proof: The theorem directly results from the description of the mechanism and protocol. The total busy window of q consecutive transmissions m_i is constructed from the maximum network latency of these transmissions $(q \cdot C_i^+)$, the latency $(3q \cdot C_{i,msg}^+)$ of control messages for each transmission (*reqMsg*, *ackMsg*, *relMsg*), plus the maximum latency resulting from preemptions of transmissions with lower priority and arrival of requests for transmissions with a higher priority.

Each $\eta_i^+(\Delta t)$ transmissions t_i must preempt maximally $\eta_i^+(\Delta t)$ times other transmissions belonging to $lp(i)$. It may also happen that, in the worst-case, the maximal number of activations of tasks from $lp(i)$ is lower than $\eta_i^+(\Delta t)$, therefore one should select the minimum from both. Each preemption request introduces an additional delay resulting from the synchronization protocol. Firstly, the higher-priority request must wait until the control message (*blckMsg*) will arrive to the appropriate client. In this analysis, the longest possible time of *blckMsg* for all transmissions belonging to $lp(i)$ is conservatively.

Additionally, each preemption from the task with higher priority is defined by the time necessary to send an *ackMsg* and receive a *relMsg* from the higher-priority application as well as latency of the *resMsg* and the maximum network latency of the higher-priority transmission t_j. Finally, in order to construct the worst-case scenario, it is assumed that requests from all streams arrive exactly at the same moment. Therefore, it is a conservative overestimation of the actual interference. Note that the assumed static priority preemptive scheduler is a special case of the general SPP scheme analyzed in [121]. ■

Dynamic Priority Preemptive Scheduler

As discussed in Chapter 3, if there is slack left for safety-critical senders - after considering the synchronization protocol (blocking and overhead)- it also may be used for improving performance of best-effort senders.

In the dynamic priority approach, BE traffic is allowed to use a path (or parts of it) of a HRT sender, when the HRT sender is not active. As soon as the HRT starts a transmission, the BE traffic is detoured or blocked, such that the HRT experiences no interference from (this) BE traffic while sending data. Hence, for each transmission of a sender i (which can contain multiple packets), the control layer must enforce detouring of any BE sharing a path, which is handled by a detouring request. Each detouring request to BE senders introduces an additional delay resulting from the change of the system state. Firstly, the critical sender must send the request message to the RM (*reqMsg*). Later, the RM blocks (or detours) BE senders. To do this, it injects the control message ($cfgMsg$) and waits until it arrives to the clients supervising BE senders. This time can be obtained with standard network analysis, deriving the worst-case latency of control messages. Secondly, the RM must ensure sufficient separation between the blocked (or detoured) BE sender and the HRT sender, i.e., ensure that all packets from the BE sender leave the NoC before HRT starts sending. This can be conservatively ensured by selecting the maximum latency of any BE on the path. Hence, this work assumes that the slowest BE sender injected a packet the same moment the control message arrives and one has to wait until this packet leaves the network. With

this, it is possible to define the additional delay that $\eta_i^+(\Delta t)$ requests from a HRT sender may experience as:

Definition 11 *The Protocol Overhead defines the additional delay which a single HRT transmission (possibly containing multiple packets) from a HRT (and/or SC) application i may experience in the worst-case due to concurrently running BE senders on path h. It can be upper bounded by [77]:*

$$O_i^{DP}(h) = Inj_i^{ctrl}(|BE(h)|) + 3C_{i,ctrl}^+ + \max_{\forall j \in BE(h)}(C_{j,ctrl}^+) + \max_{\forall j \in BE(h)}(C_{j,pckt}^+) \quad (4.13)$$

where $BE(h)$ is the set of BE senders sharing the same path h as the critical transmission i, $C_{j,ctrl}^+$ denotes the maximum latency of the control message to the BE sender j, $R_{j,pckt}^+$ denotes the maximum latency of a single packet belonging to the BE sender j, $|BE(h)|$ the number of BE senders sharing path h, and $Inj_i^{ctrl}(m)$ the time to inject m control messages. Note that for a network supporting multicast, it is possible to neglect the injection time as a single message must be sent, whose delay is fully accounted for in $C_{j,ctrl}^+$. $3C_{i,ctrl}^+$ is a protocol overhead of the RM (reqMsg, ackMsg, relMsg).

Consequently, the proposed mechanism safely blocks (detoures) BE applications whenever all critical senders belonging to the synchronization scenario have enough slack. Therefore, it satisfies the requirements when accounting for the overhead:

$$\begin{aligned}
&(r1_{ALD}) \quad D_i(1) \geq R_i^+(h) + O_i^{ALD}(h), \\
&(r2_{ALD}) \quad D_i(q) \geq R_i(q,h), \\
&(r3_{ALD}) \quad \bar{D}_i(q) \geq l_i^+(q,h) + \left\lceil \frac{q}{n_i} \right\rceil \cdot O_i^{ALD}(h),
\end{aligned}$$

where n_i is the number of packets in a transmission of stream i and hence $\lceil \frac{q}{n_i} \rceil$ the number of transmissions needed for q packets. In case of detouring BE, the set of interfering streams with the same priority $EP_i(h)$ for sender i only contains other safety-critical senders sharing path h and no best-effort traffic when deriving R_i^+, R_i, and l_i^+. And depending on the implementation of the control layer, the set of senders with higher priority might contain the control messages of other senders. Note, that the needed requirements or combination of requirements depend on the system design (i.e. not all requirements need to be satisfied in all cases).

4.1.2 Simulation

Simulation can be applied at all abstraction levels. This feature is particularly useful during the design of the control layer which can be carried out using a transaction-level modeling (TLM) approach. This method separates the design of the communication protocol (e.g. details of communication, channel, number of messages, scheduling policy of the RM and client, mapping of the RM) from the design and implementation of functional units (clients and the RM) or the communication architecture (e.g. router and network interfaces). An application of the TLM method permits to quickly evaluate and explore multiple design options through transaction abstractions used instead of a simulation of individual bits and cycles. By doing so, it is possible to accelerate the design process and to decrease modeling efforts. However, to achieve these goals, TLM sacrifices temporal and functional accuracy in comparison to modeling on the register-transfer level (RTL). Therefore, input from these tools should be used at an early stage of the NoC's design to narrow the spectrum of possible choices. Later, it can be supported by evaluation of RTL-level prototypes which although accurate are complex in terms of modeling efforts and slow in execution.

The incorporation of the control layer can be done through an extension of existing TLM simulation tools for NoCs, see Table 4.1, as it is independent of the underlying NoC architecture (cf. Chapter 3). The rest of this section describes the simulation library designed for these purposes using the Omnet++ framework. The description of the simulator designed for supporting the implementation of the control layer is structured as follows: Subsection 4.1.2 briefly presents the available simulation environments for NoCs and motivates the choice of Omnet++ and Hnocs [15] library. Next, Subsection 4.1.2 presents the model of the underlying NoC architecture, derived from the Hnocs [15] library and highlights introduced extensions e.g. additional physical layers necessary for transmissions of control messages. Finally, Subsection 4.1.2 introduces models for simulating the control layer and its main features. The evaluation of the different protocols using the introduced simulator is presented in Chapter 5.

Background and Related Work

Since NoCs have emerged as a new on-chip interconnect paradigm for multi- and many-core systems multiple research and commercial efforts focused on the development of tools for NoC-simulation and modeling. These efforts are stimulated by the size and dimensionality of the design space. The designer must simultaneously consider multiple architectural features: topology, routing, congestion control, admission control, link bandwidth, size and number of buffer queues etc.

Simulator	Framework	Topologies	Open Source	Hetero -genity	Synchronous / Asynchronous
BookSim [70]	C++	mesh /torus/trees	Yes	No	Synchronous
Noxsim [32]	SystemC	All	Yes	No	Synchronous
Nigram [88]	SystemC	All	Yes	No	Synchronous
Wormsim [63],[99]	C++	mesh/torus	Yes	No	Synchronous
Garnet2.0 [8]	C++ (gem5)	All	Yes	Yes	Synchronous
Sicosys [117]	C++	Mesh/Torus	Yes	No	Synchronous
Nostrum [94]	SystemC	Mesh/Torus	Yes	No	Synchronous
HNOCS [15]	Omnet++	All	Yes	Yes	Both

Table 4.1: Comparison of the open source NoC simulators.

The comparison of the considered, popular NoC simulation tools is presented in Table 4.1. The implementation of the control layer can follow in any of these simulators, as it is mostly independent of the underlying NoC architecture cf. Chapter 3. As the choice is arbitrary, the selection process is mainly defined by functional properties of the environment e.g. its flexibility, code re-usability, popularity among research community and industrial partners as well as availability of libraries.

For the purpose of initial design of the control layer, the simulation environment has been implemented through extension of the HNOCS library for the Omnet++ simulation framework. OMNeT++ is an object-oriented, modular, discrete-event network simulator. It has gained widespread popularity as a network simulation platform in the scientific community (more than 200 OMNEST/OMNeT++ related publications per year [111]) as well as among industrial partners. OMNEST [111] is the commercially supported version of OMNeT++.

OMNeT++'s environment provides several important advantages which can be applied for modeling of the NoC. Through extensible, modular, component-based C++ simulation framework, it offers easy integration of many independent libraries and modules i.e. different traffic sources, integration of on-chip and off-chip traffic streams, portability and flexibility of the designs. This allows to significantly reduce efforts required for modeling the domain-specific functionalities such as support for sensor networks, wireless ad-hoc networks, Internet protocols, performance modeling, photonic networks, etc.. All these libraries are developed as independent projects. OMNeT++ offers an Eclipse-based IDE, a graphical runtime environment, and an interface for hosting other tools. Additionally, this toolset

can be enhanced with extensions for real-time simulation, network emulation, database integration, SystemC integration, and others.

Consequently, Omnet++ has been used for the implementation of the open-source HNOCS library (Heterogeneous Network-onChip Simulator) [15]. Later, in the scope of the presented work, this library has been enhanced and extended for the transaction level modeling of the control-layer. HNOCS offers models of heterogeneous NoCs with variable link capacities and number of VCs per ports, following the baseline design from [37]. The HNOCS simulation platform provides modular, scalable, expandable and fully parameterizable framework which includes three types of NoC routers: synchronous, synchronous with virtual output queue (VoQ) and asynchronous. Therefore, it offers support for modeling heterogeneous NoCs, frequently applied in industrial practice e.g. FlexNoC [11] which is missing in other simulators (cf. Table 4.1),.

For supporting evaluation and verification, the HNOCS library offers a high number of tools for measurements at different granularity levels (packet or flits) e.g.: end-to-end latencies, throughput, VC acquisition latencies, transfer latencies, etc. The next section, describes the underlying NoC architecture used as the basis for the introduced mechanism.

Simulated NoC Architecture

The simulator is built using as a basis the works from [37] and [15] which introduce NoC implementation, following the principles of a performance optimized NoC, cf. Chapter 1. Therefore, all traffic is treated equally, and fair local arbitration is done independently in routers. The library provides modules to model an interconnect of the underlying NoC-based many-core system, including processing nodes, routers (incl. virtual channel allocators, ports, schedulers), and NI (sink and sources). The simulation is event-based.

The building elements of this NoC are presented in Figure 4.3. The presented NoC architecture is based on a 2D mesh with *routers* connecting individual *processing nodes*. All elements have a modular and hierarchical design. The router is composed of an adjustable number of ports, in this case five, which are fully interconnected. Each port consists of four basic modules: *inPort*, *vcCalc*, *opCalc* and *scheduler*. *inPort* is responsible for receiving arriving packets and queuing them in different buffers for different VCs. *opCalc* and *vcCalc* assign output port and output virtual channel for waiting packets, whereas *scheduler* decides the order in which packets get access to the port while leaving the router[4]. Modules are inter-changeable. For instance, the extension of the simulator proposed in this work supports round-

[4]For interested reader, the detailed description of the internal router working is available in [15]

robin based (similarly to iSLIP) as well as static priority based (preemption's of flit level) arbiters. Additionally, routers apply winner-takes-all scheduling i.e. all flits of winning packet must be transmitted before the next packet can be scheduled [37].

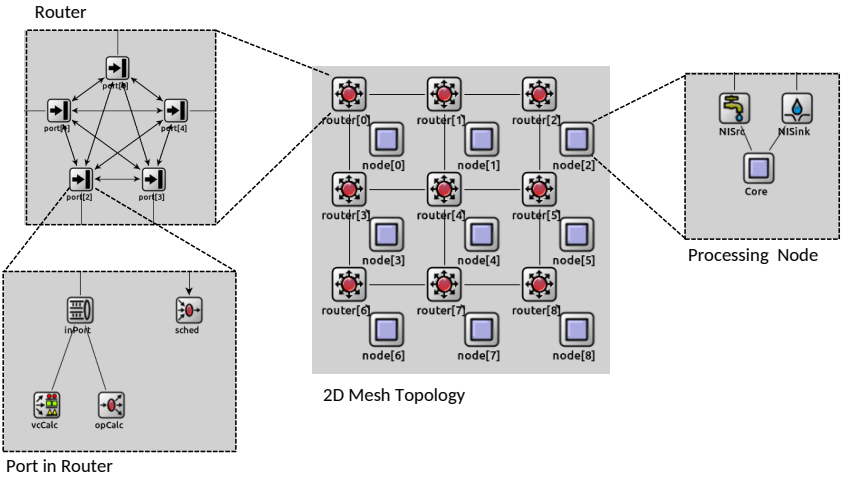


Figure 4.3: 3x3 mesh NoC with internal structure of: Router, Processing Node, Router and Port.

Each processing node is constructed from a network interface built out of two primary modules, *NISrc* and *NISink*, as well as from the processing core. The core contains a traffic generator which injects packets to the NoC. For each core, there are two main configuration parameters: the destination and packet inter-arrival timing parameters. The destination can be pre-defined (deterministic) or randomly selected using one of the OMNeT++ generation functions (e.g. uniform or exponential). Similarly, the inter-arrival times can be: randomly distributed (e.g. exponential, constant), driven by a mathematical model or defined by a trace file. Mathematical models can be applied to simulate the following traffic patterns: i) periodic activation with jitter ii) periodic burst with jitter iii) Bayesian probability defining next occurrence of periodic activation and iv) Markov chains for modeling of sporadic bursts which could happen on processors with caches. The trace based execution uses files which contain specific information about times when packets must be injected and the length of the transmissions.

The collection of data is done on the transaction level (single flit, packet latency

or round trip times) as well as for the whole logical transmission constructed of multiple packets. In addition to latency and throughput measurements, information about queue occupancies are available e.g. worst-case backlog. This allows a quick identification of bottlenecks and saturation points in the NoC.

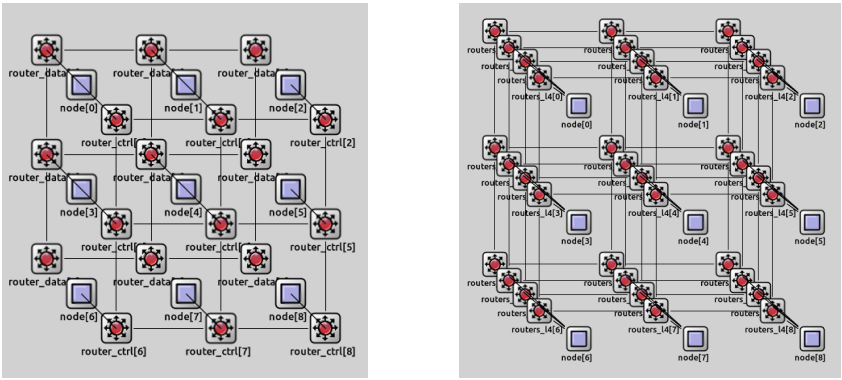


Figure 4.4: Multilayer NoCs modeled with simulator: a) heterogeneous double layer, b) homogeneous multilayer.

Finally, besides regular 2D mesh architectures, the developed simulator offers support for multi-layered chips, see Figure 4.4. This includes simulation of heterogeneous layers e.g. different routers for transmission of control and data messages as in MPPA architecture from Kalray, see Figure 4.4-a), as well as modeling of homogeneous chips constructed from multiple layers with the same router architecture, similarly to Tile64 from Tilera [151], or [73], see Figure 4.4-b).

Modules for Modeling of the Control Layer

Figure 4.5 presents the modules introduced to model the control layer. A processing node with client, i.e. *client node*, is constructed from two sub-modules: a traffic generator and an actual client logic belonging to the control layer. The working of the former module has been already discussed in the previous section. The working of the latter will be described in the scope of conducted operation, see Figure 4.6. Whenever a sender is trying to start a transmission, the traffic generator initiates a transInitMsg event (a). This message arrives to the client module and is trapped. The corresponding client sends a request message to the RM to obtain access to the network (b). After receiving the ackMsg (c), the communication may

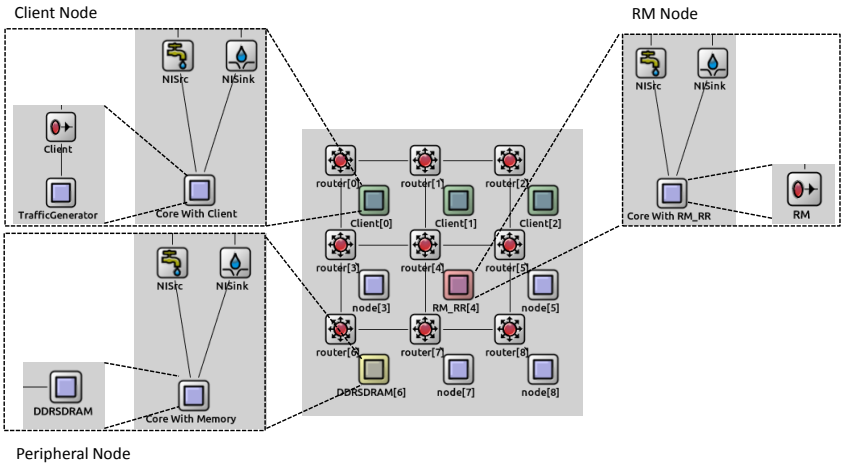


Figure 4.5: 3x3 mesh NoC with internal structure of: Router, Processing Node, Router and Port.

start (d). Once granted, the connection holds until the end of the transmission or the abortion through the client based on a predefined timeout used to prevent unbounded connection times. When the client detects the end of the transmission (e), based on the injection of the last flit, it issues a relMsg to the RM (f).

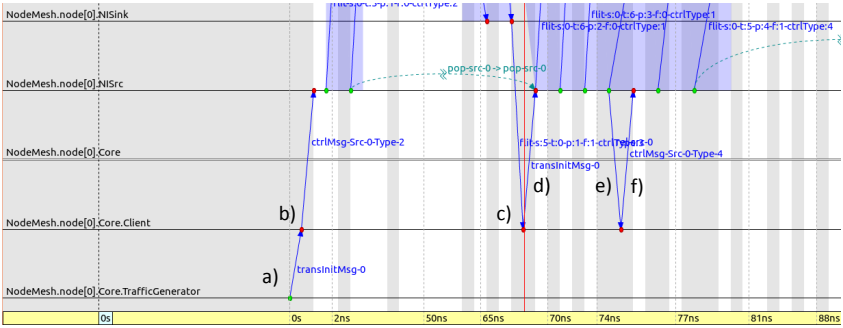


Figure 4.6: Client transmission process example (Omnet++ event log, linear time scale).

Figure 4.5 depicts the design of the processing node with the RM logic. Due

to the modular design, the arbitration unit supports all schedulers described in Chapter 3, among other round-robin, static and dynamic priority based arbitration, detouring of transmissions (multipath-scheduling), and finally adaptive rate-control. The working of the RM unit will be described in the scope of conducted operations, see Figure 4.7.

Upon its arrival, each control message from the clients is placed in the queue in the RM (a). If the queue is empty, the RM must wait for a new request to arrive. Otherwise, the scheduler decides which request from the queue should be served first (b). Each RM is sequential and serves one request at a time. The selected request is removed from the queue and starting at this moment the resource is considered to be occupied. After that, the RM must notify the sender selected for service with an acknowledge message (b). Once granted, the connection holds until the end of the transmission or the abortion through the client. From the moment of arrival of the release message, the RM considers the resource to be free again (b).

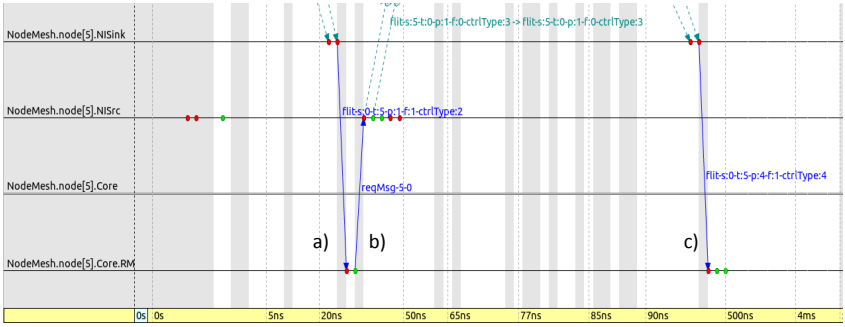


Figure 4.7: RM unit working example (Omnet++ event log, linear time scale).

Finally, the simulator is capable of modeling direct communication between applications (core-to-core) as well as accesses to memory modules. Figure 4.5 depicts a peripheral node with an interface to the DDR SDRAM. The simulator can model both write and read accesses. In the latter case, the activation of the next access to the NoC may be scheduled only after receiving the answer, i.e., self-throttling effect [37]. This is especially useful for modeling traces of the on-core execution where response time of an application depends directly on the interference during accesses to the globally shared resources. Modeling of the response time of peripherals can be done twofold: i) by using a static worst-case delay ii) by introducing a peripheral module to model the actual behavior e.g. state dependent response time.

4.1.3 Tool for Protocol Configuration

The initial design phase done with the analysis and transaction-level simulation allows the designer to evaluate and verify the protocol architecture and the selected method for resource arbitration. This assessment is done on a higher abstraction level (e.g. transactions and contracting) for the purpose of an early exploration of a variety of inputs and possible strategies for resource assignments. As a result, the set of constraints for the low-level design must follow. Examples for the characteristics are: number of synchronization scenarios, maximal and minimal frequency and number of initiated transmissions, states of RM, size of messages as well as upper limits (i.e. deadlines) for transmission latencies, which do not violate the safety of the system. Consequently, a tool can be provided which makes this process (semi) automatic and generates the code of the clients and RM depending on the selected and evaluated resource allocation scheme.

Such tool for (semi) automatic generation of the IPs required for the deployment of the control layer could in turn extend the capabilities of other EDA toolchains, e.g., tools from Synopsys and Xilinx. The automated design flow, including the design evaluation and enhancements, would introduce optimized real-time solutions without the need for specialized know-how from the designer. It is critical for enabling "safety as a feature" of the future NoC designs and simplifying the process of the system's verification.

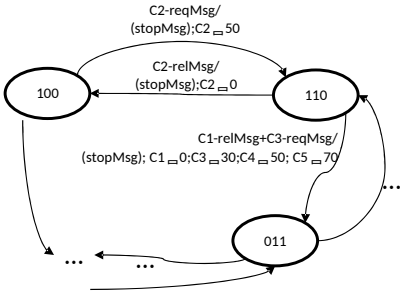
States of the System

Note, that the concrete implementation of the protocol strongly depends on the selected arbitration method and capabilities of the underlying NoC architecture. In this section a generic, exemplary deployment supporting a combination of the mechanisms from Chapter 3 is used. The selection of features for control layer was done to expose the reader with a multiplicity of possible options and at the same time to avoid an unnecessary complexity which could make understanding difficult. However, the presented solution is general enough to be straightforwardly extended for purposes of the particular implementation. Consequently, the protocol considered in this section supports the following features: priorities, rate control and multiple paths per senders (adaptive load distribution).

Different sets of running senders and resulting settings for rate controllers define independent states of the system, as shown in Figure 4.8. With the profile and requirements of each application, the analysis can be used to obtain the set of applications that are allowed to use the network simultaneously and to identify the necessary settings for rate controllers. Later, these senders must be synchronized with the control layer.

Active Senders	Settings				
	HRT Senders			BE	
	C1	C2	C3	C4	C5
100	50	0	0	0	0
110	50	50	0	0	0
...	..	..	..	..	..
011	0	50	30	50	70
...	...				
010	0	50	0	50	100

a)



b)

Figure 4.8: Control Logic of the RM including the tables with synchronization scenarios a) and the state transition graph used by the arbiter b), based on [84].

Figure 4.8 presents a tabular description of settings (left) for different states of the system. It also contains an exemplary state machine (right) which defines the main workings of the RM control unit. In this scenario there are three HRT senders (C1 - C3) and two best effort senders (C4 - C5). The state of the system solely depends on the HRT applications - three bit representation on the left. In the first state, C1 is running alone and requires 50% of the bandwidth. The second line shows the case where the senders C1 and C2 are active and get the same share of the network (50% each), i.e., symmetric guarantees. In the fourth line, two senders (C2 and C3) are trying to use the NoC with different requirements on the bandwidth (50% and 30%), i.e., asymmetric guarantees. Moreover, C2 is sharing a link with the BE sender C4 which may use the remaining 50% of the bandwidth. Similarly, the BE sender C5 may only use 70% of bandwidth as it shares a link with C3. In the last case only one HRT sender C2 is running. As before, C4 is allowed to use 50% of the bandwidth. Additionally, C5 receives the full bandwidth, while C3 is not active and it does not interfere directly with C2. During run-time, the RM uses the current self-image of the system, i.e., information about running senders and occupied resources, to dynamically adapt the NoC to the changing workloads. The arrival of a request or release message triggers a change of the system state. Note that the transitions between states from the table are triggered only by critical senders and not by BE. For instance, if C1 is active (state *100*) and a *reqMsg* from C2 arrives a transition to the state *110* will follow. Moreover, if there are multiple outstanding requests or releases, they can be handled at the same time (e.g. if a sender has to wait for another sender to finish). For example, if the NoC is in state

110 and there can be two messages (relMsg from C1 and reqMsg from C3) the RM can conduct directly the transition to state *011*.

Specification of the Protocol

Table 4.2 presents exemplary content of the Look-Up Table (LUT) used by the RM. Synchronization scenarios (states of the system) are defined by sets of applications which are allowed to use a NoC concurrently. The scenarioID must be unique and is defined by a binary vector where each bit identifies a sender (1 - active, 0 - deactivated) and a priority (lower number identifies higher priority). The change between states is triggered by the arrival of a request or release messages from a HRT sender. States with the highest priority are selected first. If there are multiple states with the same priority they are treated equally i.e. round-robin arbitration is used. Exemplary settings for senders are presented in Table 4.3. For each state and sender, the rate control settings are used. For instance, the maximum number of packets allowed by a time period T - max counter (cnt_max) and the path which should be used by the arbiter. Finally, Figure 4.9 presents an exemplary format of the control messages. The protocol is realized with three messages: relMsg, reqMsg, cfgMsg. The type of each message is specified by its unique ID (*MsgType*), followed by the sender ID (note that on the same core there might be multiple synchronized senders). Finally, the cfgMsg contains payload allowing client identification of settings for a given scenario. Here two implementations are possible. The first one requires a longer message containing full configuration (e.g. route and maximal value for the counter). The second implementation introduces shorter messages, i.e., only the scenario ID is transmitted, but requires the client to store the table with settings locally.

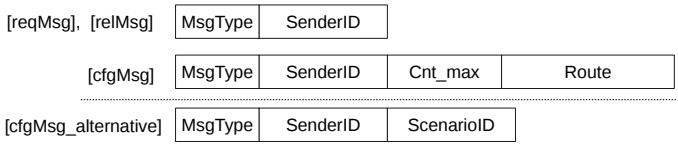


Figure 4.9: Content of the control messages used for the synchronization protocol.

The number of scenarios and the number of synchronized senders is established with the introduced earlier tools for analysis and simulation. The main trade-off lies between optimal performance (i.e. high number of states) and protocol overhead (i.e. high number of mode changes, propagation latency). Note, that some senders may be excluded from the arbitration (e.g. run with constant settings in all

ScenarioID (Active HRTs)	Priority
000 …001	1
000 …011	2
000 …010	3
… …000	…

Table 4.2: Content of a LUT defining system modes (depending on active senders) and their priorities.

ScenarioID	SenderID	Cnt_Max	Path
000 …001	000 …001	100	route_2
000 …011	000 …001	40	route_1
000 …011	000 …010	23	route_1
…	…	…	…

Table 4.3: Settings for different scenarios considering maximal arrival rates and paths for the transmissions.

modes) in order to decrease the number of state transitions in the NoC. The worst-case analysis is employed to derive the required settings for the rate controllers on each processing node, i.e., the system must assign for each HRT sender a sufficient bandwidth share to reach its deadlines and requested throughput. It also provides the upper bound on the spatial and temporal overhead e.g. maximal processing time for the control layer modules (clients and RM), size of the queues, amount of data necessary to identify scenarios, etc.

4.2 Implementation

In the previous chapters, a collection of synchronization protocols for the control layer has been proposed. This section explains how to translate requirements and constraints of the design into a working setups with the control layer. The method of implementation as well as the resource overhead resulting from the integration of components belonging to the QoS control plane must be considered in the context of a concrete MPSoC architecture and its specific deployment. For instance, clients and RM modules can be realized in form of hardware/software co-design re-using existing chip components (e.g. stand-alone software library for running the RM, software extensions of the existing OS/RTEs for adjusting parameters of NIs - clients functionalities) as well as solely in hardware (e.g. stand-alone

modules, extensions of NIs, microcoded processors). The former offers full flexibility, the latter maximum performance. Independently of the type of implementation, clients must meet the following functional requirements:

- trapping/intercepting and distinguishing between different transmissions;
- generating request messages;
- processing acknowledge messages;
- detecting configuration messages *cfgMsg* and changing the QoS settings in NIs;
- detecting the end of transmissions and releasing resources.

The actions of clients can be completely transparent to the running applications which assures compatibility with the legacy software and most of existing applications. The performance of the control layer and NoC is strongly influenced by the granularity of the synchronization i.e. how much data is necessary for a client to identify initiated transmissions. Clients may conduct a synchronization depending simultaneously on multiple factors, for instance including:

- each initiated transmission from the processing node,
- transmission using a particular set of resources (virtual channel and/or path),
- transmissions initialized by a particular task/application or module within the processing node e.g. DMA engine or a specific task running within an OS,
- transmissions targeting a particular receiver,
- based on the monitors (and/or stored by them data) e.g. continuous synchronization depending on the frequency of the initiated transmissions.

The main trade-off lies between the amount of resources (processing time, stored data, size of the client) and the precision of the synchronization i.e. a better (fine-granular) service costs more. Similarly, the RM control unit must fulfill the following functional requirements:

- receiving and qualifying control messages (type and sender);
- distinguishing between different synchronization scenarios;

- switching between scenarios according to the predefined set of rules (scheduling method);
- assuring a safe transition between system states (preventing sporadic overloads);
- and generating control messages.

In the future, the RM may be enhanced with advanced mechanisms based on monitoring, for a fully flexible deployment. However, the precision of the arbitration usually comes in pair with the implementation overheads in terms of increased resource cost and synchronization latency, as in case of clients. Moreover, a higher complexity of the RM has usually a negative impact on the formal analysis, see the discussion from Sec. 4.1.1.

The resource overhead depends directly on the complexity of the introduced arbitration (e.g. dynamic switching between off-line defined scenarios or full on-line resource allocation), verification methods and availability of existing infrastructure for a particular configuration of a chip. However, in many NoCs this cost can be amortized by the re-usability of existing NoC components e.g. monitoring infrastructure in NIs. For instance, as the complexity of the central RM unit is higher than a client, it can be implemented in the form of a stand-alone processing node. In order to do this, the designer may use a supervisor "high-performance" nodes available in many MPSOCs e.g. IDAMC [140], MPPA [40], FelxNoC [11]. Similarly, in many NoC architectures, NIs are already equipped with interfaces for the configuration of admission control settings e.g. rate limiters. This configuration is typically done at the platform startup by supervisor nodes. Therefore, it is typically possible to directly re-use, adjust or enhance existing mechanisms.

The rest of this sub-section discusses the challenges in detail. Firstly, Section 4.2.1 introduces possible communication channels for transmitting control messages between the RM and clients. Next, Section 4.2.2 discusses the possible software implementation of the RM and clients (through maximal re-using of the existing platform elements) whereas Section 4.2.3 concentrates on the hardware architecture.

4.2.1 Transmission of Control Messages

As discussed, safety and latency of control messages have a critical influence on the performance of the QoS control plane. In general, it is possible to transmit control messages using the following methods:

- within the same NoC infrastructure as data transmissions (i.e. using data layer);
- using an additional (physical) NoC layer;
- using an additional bus, e.g. bus-enhanced NoC;
- using a separated and directed set of signal lines between the RM and clients.

The most straightforward solution is to use the same infrastructure for transmitting control messages as for data packets e.g. any available VC capable of providing latency guarantees. This deployment is presented in Figure4.10.a). In such setup, there is no need for modification of the underlying NoC. Of course, this approach, although resource efficient, may negatively impact the temporal guarantees for both latency of control and data transmissions as they are allowed to freely interfere. For bounding transmission time of control traffic routers with priority based schedulers can be used, see [83],[31], [66]. Such routers have been described in Chapter 2. For instance, control layer traffic could be mapped to the VC with the highest priority i.e. it cannot be blocked by any other ongoing data transmission. Additionally, to reduce the blocking exerted by control messages on the data layer, their payload should be preferably small e.g. *cfgMsgs* containing solely information for identification of transmissions (e.g. identification number), and the type of the message. Therefore, this solution should be introduced after examination of the trade-offs between synchronization protocol and other traffic in the NoC. This can be done by using previously described analysis and simulation tools. A higher number of issued control messages (for instance due to the size, granularity and number of synchronization scenarios) inevitably increases the introduced overhead with respect to both time and resources. To increase performance one may apply a dedicated independent control NoC, cf. Figure4.10.b). In such setups, control messages are transmitted using an independent layer of routers (as well as links or even additional NIs) so that control messages are not blocked by any other traffic. Therefore, a control layer is physically separated from the data layer. Of course, these implementations introduce a higher hardware overhead. However, the cost of such solution may not be that high as it may intuitively seem. Firstly, routers for the control layer may be much simpler than the routers in the data plane e.g. no support for multiple buffer queues in routers ports (VCs), simple single-stage scheduler or even different operational frequency. This could drastically decrease the size of the routers as proved by the related studies e.g. [73]. For more detailed discussion please refer to Chapter 2. Indeed, some of existing MPSoCs already support multi-layer NoCs e.g. Tilera Tile64 or Kalray MPPA. In

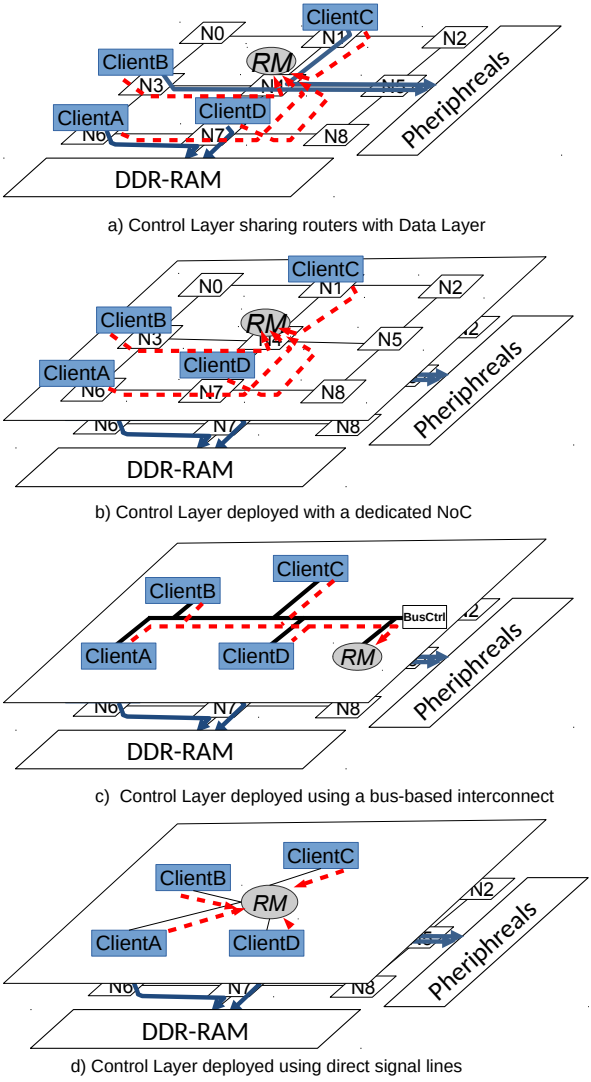


Figure 4.10: Different MPSoCs setups for transmissions of control messages.

Tile64 architecture independent layers are used for spatial separation of the traffic streams i.e. Quality-of-Service support. MPPA applies a heterogeneous architecture where simpler routers in C-NoC (e.g. shorter buffer queues in ports) are used for configurations and monitoring of the processing nodes, whereas more complex routers (e.g. larger buffer queues) in D-NOC are used for DMA based data transmissions. Moreover, many MPSoC architectures introduce an additional interconnect layer also to assure an error free execution and fast recovery/ fail-over i.e. spatial redundancy. Summarizing, in many setups control traffic can be straightforwardly accommodated and spatially separated from the data plane, while using the existing interconnect infrastructure.

Alternatively, the spatial separation of the control messages can also follow using a bus-based interconnection layer as proposed in [97]. In such enhanced designs, Figure4.10.c), a bus layer with a central arbiter is used to accommodate low latency and low bandwidth transmissions (e.g. short control messages) whereas the NoC layer is used for high throughput point-to-point connections (data plane). Such multi-layer interconnects allow decreasing hardware overhead, when compared to multi-layer NoC architectures, while circumventing some of their main weaknesses e.g. latency of control signals, complexity and cost of broadcast operations. Consequently, the bus is used for a selected subset of specialized system operations which decreases the amount of transmitted data (thus permits avoiding scalability issues) while benefiting from the proximity of neighboring nodes.

Indeed, these properties combined with a moderate number of processing nodes in many MPSoC architectures known from research (e.g. from 4 to 32 processing nodes in IDAMC assuming Virtex-6 FPGA deployment) and commercial deployments (e.g. 16 processing nodes in MPPA, 64 processing nodes in Tile64) could even motivate connections with direct signal lines between clients and RM arbitration units, see Figure4.10.d). Applying a dedicated set of direct signal lines offers a maximum performance but at the same time suffers from the well-known limitations of the centralized and direct communication i.e. does not scale well with the size of the chip and the number of synchronized senders.

Note that independently of the form of deployment, the frequency, number and latency of necessary synchronizations and re-configurations strongly depends on the use-case. In some cases, to mitigate the unwanted side effects, the designer may conduct a trade-off analysis during the design phase providing an estimation of the overhead resulting from the global arbitration. For instance, it is possible to keep settings for some applications on a constant level (without endangering their deadlines) in some or all modes i.e. decrease the number of modes of system work. This allows limiting the number of necessary synchronizations and re-configurations reducing the probability of global arbitration being the bottle-

neck of the design. Moreover, it decreases the amount of resources necessary to keep system settings by RM and clients. The predominant trade-off is between fine granular adjustments (for increasing performance) and both temporal and hardware overheads.

4.2.2 HW/SW Co-design for Clients & RM

The resource overhead resulting from the integration of the QoS control plane must be considered in the context of a concrete MPSoC with the goal of minimizing the implementation costs. Note that hybrid solutions (hardware/software co-design) re-using existing chip components are frequently used for MPSoCs and constitute an appealing alternative to the homogeneous implementations (e.g. clients and RM realized fully in hardware).

As the complexity of a central RM unit is higher than a client, it can be implemented as a software component running on a stand-alone processing node, for instance a small ARM core. This allows maintaining high flexibility (deployments with different scenarios and arbitration methods) as well as to decrease the development costs. As previously discussed, the designer may use supervisor nodes available in many MPSOCs e.g. IDAMC [140], MPPA [40] to follow these design patterns.

Similarly, mechanisms for rate control are applied in many MPSoCs and are frequently considered as their standard feature. [5]. For example, the commercially available NoCs such as Tile64 [151], MPPA [40] or FlexNoC [11], offer rate limiters (along with necessary monitors and interfaces) realized in hardware as integral modules within NIs. Therefore, the software implementation of clients can follow as a straightforward extension of this mechanism. The necessary client logic (e.g. generation of control messages, identification of the sender) is implemented in form of a software module running on the processing node. This new application is later adjusting settings of QoS mechanisms deployed in hardware e.g. rate limiters or MMUs in NI from baseline architecture described in Chapter 3.

Finally, the internal working of the RM as well as client modules can be divided between hardware and software modules. For instance, in case of the RM control and arbitration units can be deployed in software which assures high flexibility, whereas LUTs with settings for synchronization scenarios can be stored in hardware which allows to significantly decrease the time necessary to identify senders and processing of control messages. The same principles apply to clients.

[5] Recall, that admission control with late limiters in NI is necessary in both performance optimized as well as real-time NoCs. Otherwise results of backpressure and propagation of blocking can lead to the NoC saturation and endanger its performance or real-time requirements, cf. Chapter 1

As discussed, in case of a client detection of the access and synchronization can be done by the extension of an operating system and / or hypervisor whereas the rate-control and low-level monitoring are done in hardware. This allows to simultaneously decrease resource overhead (small extension of the NIs interfaces) and simplify synchronization which could happen on the higher abstraction and virtualization layers.

The discussion of the software deployment of the control layer, begins with the functional description of clients and the RM. Later, it accounts for placement of the control layer code within infrastructure of the software stack running on the MPSoC.

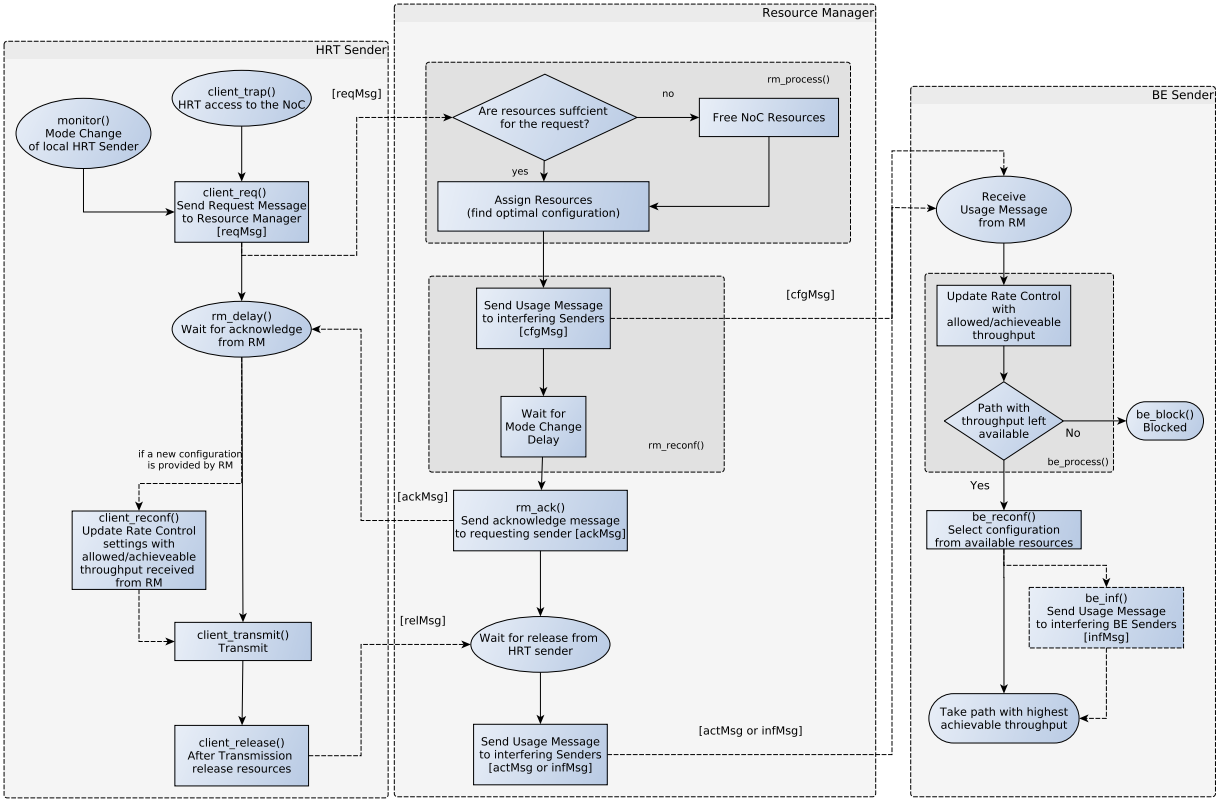


Figure 4.11: Workflow of the software modules constructing the control layer incl. RM unit, client for HRT senders and client for BE senders.

Workflow of the software modules.

The presented discussion considers a specific, but realistic, scenario where the control layer is applied in a NoC-based manycore. The workload in this platform considers traffic with hard real-time requirements (HRT) deployed along with BE applications. The synchronization protocol concentrates on the subset of most important features of the introduced mechanism assuming the deployment which follows principles of HW/SW co-design. Consequently, the RM is deployed fully in software on the dedicated core, whereas clients' libraries are running on processing nodes and are capable of controlling QoS modules within NIs. The selection of features for the control layer was done to expose the reader with the multiplicity of possible options and at the same time to avoid an unnecessary complexity which could make understanding difficult. However, the presented solution is general enough to be straightforwardly extended for purposes of the particular implementation. Implementations for other resource arbitration protocols, cf. Chapter 3, can be derived as a direct extension of the presented scheme.

Actions of the software modules are explained in the scope of the operations conducted by clients and RMs using an exemplary protocol with the workflow presented in Figure 4.11.

Step 1: *client_trap*(): each transmission from the HRT sender must be trapped (intercepted) before an application may access the NoC.

Step 2: *monitor*(): alternatively to explicit trap() function, an on-core (processing node) monitor may signalize the change of application state (e.g. faster arrival rate of NoC accesses). This change must be firstly acknowledged by the RM as it may affect other senders sharing the NoC. Consequently, after the signal from the monitor, the client traps the next arriving access and process as in case of newly established connection.

Step 3: *client_req*(): Later, the client issues a request message (*reqMsg*) to the *RM* for granting the access to the interconnect resources.

Step 4: *rm_process*(): Each RM is sequential and serves one request at a time. If there are no pending requests, the *RM* must wait for a new request to arrive. For each pending request, processing is done in the scope of three actions: evaluation, reconfiguration and assignment. Firstly, available NoC resources (link bandwidth and buffer space) must be evaluated with respect to requirements of the transmission. If there are enough resources to accommodate a new data stream, a new assignment is done. The goal is to provide an optimal allocation of interconnect resources i.e. minimize

possible interference and dependencies between senders. If there are not enough interconnect resources, the RM must assess the possibility of the re-configuration. Both assignment and re-configuration are done by selecting an available configuration from the predefined[6] set of scenarios, as discussed in previous sections. Therefore, the RM's internal working resembles a finite state machine, where state change is triggered by the arrival of the control message.

Finally, if a request cannot be granted, for instance due another ongoing communication with a higher-priority, it is stored in a request-queue. When the resource is free again and the request-queue is not empty, the *RM* selects the request with the highest priority. Later, the request is removed from the queue.

Step 5: *rm_reconf*(): If necessary, the RM must propagate the new settings to other senders running in the system, i.e. conduct re-configuration of the NoC. In the considered setup, BE senders are affected. Consequently, the RM sends to each interfering transmission the cfgMsg, defining new settings for the rate controllers.

Step 6: *be_process*(): To assure QoS BE senders must be also protected by clients. Consequently, the client must enforce the rate control on the node using the setting provided by the RM.

Step 7: *be_block*(): If there is no alternative resource (e.g. path) available for the BE sender it must be blocked.

Step 8: *be_reconf*(): Alternatively, if there are available interconnect resources, the client protecting the BE sender may reconfigure the connection to use a different set e.g. alternative path through the network.

Step 9: *be_inf*(): Additionally, in the more advanced deployments of the synchronization protocol, clients may initiate a nested re-configuration, informing other BE clients about the changes. Both, *be_reconf*() and *be_inf*() should result in selecting optimal settings for the BE sender without endangering the safety of the system.

Step 10: *rm_delay*(): Each system mode defines the set of running senders (BEs and HRTs) as well as the maximum rate of initiated transmissions, i.e. the max-

[6] Note, that self-aware online re-configuration is theoretically possible, but constitutes an orthogonal problem and therefore is left outside the scope of this work.

imum possible interference that a transmission can suffer from other active senders. Consequently, if the transition between different scenarios happens (e.g. activation of a new sender) without the supervision from the RM, the packets from transmissions initiated in different system modes may interfere. This can cause sporadic overloads and transient load effects, which may endanger the system's safety. Therefore, the safe switching between different frequency levels, i.e. safe mode changes, is of the fundamental importance for the real-time properties of the system and must be safely enforced by the proposed control layer.

Consequently, after initiating the reconfiguration (*rm_reconf* function) the RM must delay the acknowledgment until the packets initiated in the different system mode leave the NoC.

Step 11: *rm_ack*(): The *RM* notifies the appropriate client with the *ackMsg* which unblocks the granted transmission. From this moment, the resource is considered to be busy once again.

Step 12: *client_start*(): After receiving the *ackMsg*, the transmission may start. Once granted, the connection holds until the end of the sender's transmission or the abortion through the client based on a predefined timeout to prevent unbounded connection times.

Step 13: *client_rel*(): When the client detects the end of a transmission (e.g. based on its time-budget/timeout or injection of the last flit), it issues a *relMsg* to the appropriate *RM*.

Step 14: *rm_rel*(): As soon as the *relMsg* arrives, the RM considers the resource to be free again. If there is a pending preempted transmission with lower priority, the *RM* resumes its execution after sending a *resMsg* to the appropriate client. Otherwise, a new request is selected from the queue.

The discussed steps present the set of actions which software modules must implement in order to introduce the proposed control layer. However, these modules may have additional placements within the run-time environment and/or software stack influencing, both, design overhead and their capabilities. The incorporation of the main RM-unit realized in software is relatively straightforward as it defines a self-contained and independent functionality. For this purpose, the designer may use one of the processing nodes or dedicated management cores frequently available in MPSoC for maintenance , e.g. boot-up, fail-over or monitoring.

4.2.3 Hardware NoC-Extensions

This section provides an overview of a *hardware design* required for the implementation of modules belonging to the proposed control layer. The control layer deployed fully in HW inherently introduces the low-level approach. The most intuitive design choice is to implement the client as an extension of the Network Interface connecting processing node with the NoC. Next, the RM can be introduced as an independent and dedicated node for minimizing the temporal overhead. Note that, both units can be implemented as an independent hardware modules controlling accesses before they arrive at the network interfaces (NIs). Moreover, due to the low-level approach the actions of the supervisors and protocol layer can be completely transparent to the running applications, thus compatible with the legacy software, i.e. there is no need for costly code adjustments.

The motivational example with such deployment is presented in Figure 4.12, where a NI unit is enhanced with a dedicated "state" module for the realization of the client functionality. This module contains information about active senders running on the processing node and their runtime properties, i.e. active connections, deadlines, rate limiter settings. Moreover, the client must contain the control logic to connect it to a standard packetization/depacketization interface for issuing and receiving the control messages. The actions of the sub-modules belonging to the funcionality are explained in the scope of steps which must be conducted in the process of transmission handling.

Whenever a sender running on a node is trying to access the NoC, the client will detect and trap (intercept). Later, the client must identify this access, i.e. distinguish between different connections/transmissions. This is done by the *state module*, which is equipped with a lookup table (LUT) with an entry for each transmission requiring synchronization and the state of the possible resources (e.g. paths) informing if they are currently available. As discussed before, the state depends on the current active senders and shared links. The amount of information stored by the state module strongly depends on the version of the implemented protocol. In some scenarios, it may contain a simple register defining on or off state of the NI. This functionality may be straightforwardly extended by providing the setting for the rate controller, i.e. maximum rates for initiated transmissions depending on the system mode. The size of the LUT depends directly on the number of different connections and their parameters (connection settings and system modes) as well as senders requiring synchronization. As in case of the software deployment, the trade-off lies between the fine granular adjustments for improving performance and necessary resources. Note, that in case of more complex synchronization protocols for the control layer (e.g. adaptive load distribution from Section 3.6.4), each

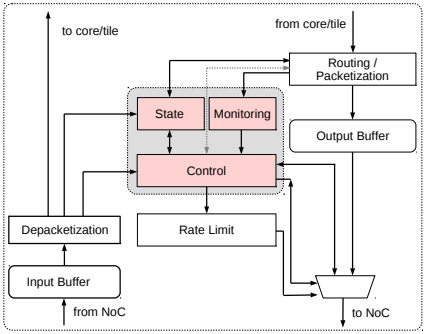


Figure 4.12: Simplified structure of the client logic implemented in form of an extension of the NI.

sender may use several paths and on each path various rate controller settings.

Similarly to the software implementation, the settings for different modes may be stored in the client or in the RM. In the former case, the client's area is larger but the protocol overhead (size of transmitted messages) is lower. In the latter case, the RM must send to the client correct settings (and not only ID of the appropriate scenario) each time. This additional payload inevitably increases the overhead for other running transmissions. Based on the information and settings from the LUTs, the RM control module decides if the access can be granted using available resources and if an additional negotiation with other senders through the RM is necessary. If the negotiation is required, the control module generates the corresponding protocol messages and must ensure that the network access is delayed until the negotiation is finished i.e. safe mode changes cf. Chapter 3.

The optional monitor module can be used to transparently detect workload changes for the applications deployed on the particular node (e.g. variations in the accumulated frequency and length of the initiated connections). To achieve this objective, the module can monitor the access behavior of the sender, e.g. compare the inter-arrival time of requests [29, 65]. The exemplary deployment of the client's LUT is presented in Figure 4.12. It follows in a system with a memory mapped NoC, where NI introduces transparent address translation. In the considered setup the local physical address (for the respective processing node) is converted into a remote address (on another processing core) and forwarded using the NoC. This releases the senders (which are executed on the processing nodes) from having knowledge about the operation of the interconnect (e.g. correct routing and ad-

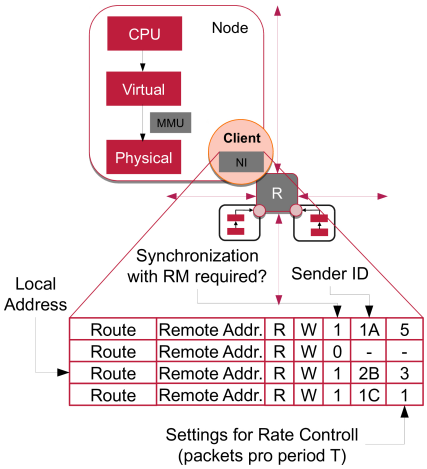


Figure 4.13: Synchronization data for client stored in address translation tables for address mapped NI.

dressing). The actual translation (including proper NoC addressing) is done by the configurable address translation module in the NI. This module is equipped with additional tables containing information about the route and destination (remote address) for a particular transmission. The implementation of the client is done through the extension of this mechanism. Firstly, a new column is added to assess if the synchronization with the RM is necessary. Next, information about the sender's ID must be added for identification of the synchronization scenario (assuming multiple of them). Finally, QoS settings are stored in form of a counter for rate limiter defining the maximum number of packets injected by the particular sender/node per a base time period. Note, that the last field can be omitted at the cost of the higher protocol overhead, i.e. control messages may contain settings, but they will be longer.

The RM can be implemented in the form of a stand-alone processing node or built into one of the existing network modules. Figure 4.14 presents a block-diagram of an exemplary RM deployment. It is possible to distinguish the four major components: 1) qualifier 2) message queue 3) Look-Up Tables containing synchronization scenarios and finally 4) control module (i.e. resource arbiter). The fifth module for monitoring is optional and will be discussed at the end of the section.

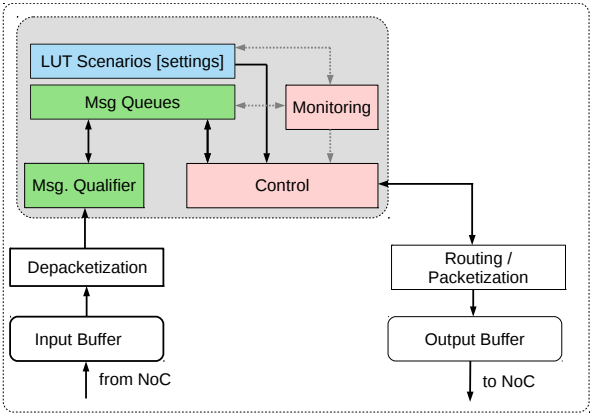


Figure 4.14: RM unit implemented in hardware as an independent module connected with the NI.

Firstly, the qualifier module must detect the arrival of a new ctrlMsg. This requires payload decoding for classification. Later, messages are stored in queues. The number of queues depends on the protocol implementation. For example, there may be a separate queue for each of the message types, i.e., one for requests and one for releases or by sender and message types implying the predominant trade-off between processing speed and amount of necessary resources. Next, the RM must be equipped with the control module which is responsible for granting, preempting and resuming transmission as well as releasing resources. The settings for different modes of the system are stored in the LUT module. Finally, the decisions from the control module are issued in form of ctrlMsgs and converted into NoC packets.

Additionally, the RM may include a monitoring module which observes system events: the frequency of activations, load, frequently used scenarios, failures. Later, this monitoring module may adjust the control settings appropriately. The scalability of the system can be increased through the implementation of multiple RMs protecting separate synchronization scenarios (regions of the NoC). Such synchronization decreases the average transmission time of control messages and distributes them spatially.

4.3 Summary

This chapter discussed the workflow and tools necessary for the design, formal verification and configuration of a NoC-based system using a control layer. Firstly, Section 4.1.1 has shown that the resulting design relies on the correctness of the models for the underlying NoC's hardware and application's behaviors. Therefore, safe implementations demand inputs from the data layer to provide admissibility tests. In each system's state, the low-level arbitration in NoC (arbiters in routers) as well as the maximum load from processing nodes must be predictable. Achieving these goals in most contemporary NoCs is fairly straightforward and requires:

- the identification of interfering senders on every path (according to the routing algorithm and mapping)
- a temporal synchronization of transmissions competing for shared resources.

Additionally, these two actions are separated and designed independently in order to enforce a predictable behavior [110]. Consequently, the creation of analysis models usually does not require any special hardware NoC extensions. Indeed, the recent advances in the real-time analysis domain ([89], [142], [44]) are proof that even standard NoCs, with complex two-staged arbitration (e.g. iSLIP [101]) and virtual channels (VCs), are efficiently analyzable allowing to compute the worst-case network latencies if the aforementioned traffic regulation and static routing is applied.

If the underlying NoC architecture complies with the requirements resulting from the need for formal verification, the designer may start considering the verification of requirements necessary for the implementation of the extensions for the control layers. The proposed tools for analysis and simulation have been introduced in Sections 4.1.1 and 4.1.2 respectively. The main goal is to provide a set of requirements for the implementation of the control modules. These actions are discussed with respect to considered elements of the control layer. Section 4.1.3 discussed the translation of these findings into the concrete settings and interfaces which are necessary for the implementation of the control layer. Recall, that this process of configuration can be automatized allowing extensions of the EDA tools.

The design of the control layer (RM,client as well as synchronization protocol) has two primary requirements: the required processing time and the amount of required hardware resources. The deciding factors are the complexity of the protocol and the size of the design. Both are interconnected, i.e. along with an increasing complexity of the main central unit usually comes an increase in the time necessary for the processing of a single request. The complexity is defined by the

three factors (same for RM and clients): (i) number of synchronized senders, (ii) frequency (granularity) of conducted synchronizations and (iii) the type of arbitration. The analysis of the protocol with respect to these factors with the introduced EDA tools should provide the designer with a set of baseline system constraints e.g. the maximal length of the queues for incoming messages, the desired average and worst-case performance of the central unit and the number of system states which must be handled by the manager (e.g. amount of data which must be stored to define settings for a particular sender in a particular state).

The implementation of clients and RM is discussed in Section 4.2. The resource overhead resulting from the integration of the QoS control plane must be considered in the context of a concrete MPSoC. For instance, clients and RM modules can be done in software (e.g. stand-alone libraries, extensions of the existing OS/RTEs) as well as in hardware (e.g. standalone modules, extensions of NIs). The former offers full flexibility, while the latter maximum performance. Note that hybrid solutions (hardware/software co-design) re-using existing chip components are also possible.

Finally, there is a requirement of the backwards compatibility i.e. the design dilemma if the synchronization with a client should happen transparently to the running applications or after their modifications. The former may increase the complexity of the clients as well as enforce certain limitations with respect to the identification of initiated transmissions. The latter may allow to improve the clients capabilities and simplify their structure, but requires adjustments of the legacy code, which can be expensive or sometimes even impossible (e.g. licensing).

The predominant requirement for the control messages is the *worst-case and average latency* of the transmission i.e. the maximum length of the transmission from the client to the resource manager. This constrain directly influences the temporal overhead resulting from the synchronization protocol, thus performance of synchronized senders. An additional requirement is the *amount of data* which must be transmitted as it has a severe impact not only on the transmission latency but also on the other important properties of the NoC architecture e.g. consumption of dynamic power necessary for conducting the transmission, overhead introduced by the control traffic on other senders, complexity of message creation/processing.

The introduced design-flow offers benefits for both SoC manufacturers and OEMs integrating highly dynamic real-time applications. The former may extend applicability of their NoCs to the real-time and/or safety-critical domains at the low cost e.g. software implementation re-using existing components, whereas the latter can use already available products profiting from low-prices resulting from high volume production, i.e. no need for custom QoS-oriented ASIC designs.

Chapter 5: Evaluation

The experimental evaluation presented in this chapter explores benefits as well as design trade-offs resulting from the implementation of the control layer. The investigation concerns the obtained worst-case guarantees as well as the average performance in realistic setups and synthetically generated corner cases. The former is done with the simulation framework introduced in Section 4.1.1, the latter is performed using the simulator from Section 4.1.2. Additionally, two test deployments with existing many-core platforms have been considered: the commercially available MPPA [40] and the research oriented IDAMC [140].

The rest of this chapter is structured as follows: Section 5.1 presents a detailed description of the experimental setup; Section 5.2 contains the analysis of the worst-case behavior in systems deploying the proposed solution whereas Section 5.3 analyses the average performance; Section 5.4 evaluates the introduced resource overhead for the control layer and finally Section 5.5 presents the two case-studies mentioned above carried out with existing MPSOC architectures.

5.1 Experimental Setup

In this section, we describe the setup used in the experiments. The NoC architecture follows the baseline design from Chapter 3 which is realized with tools from Chapter 4. The first two groups of experiments, from Sections 5.2.1 and 5.2.1, are designed to test the assumptions from Chapter 3 with a special focus on design trade-offs. These experiments consider the following sets of benchmarks: synthetic traffic generators, memory trace-based processor models and models of real-application (two different use-cases). A detailed description of the used workloads is presented in Section 5.1. Other details regarding the configuration of the baseline NoC architecture from Section 3.1 are presented in Section 5.1. Later, in a consecutive experiment series the arbitration of these NoCs has been adjusted to test different QoS mechanisms, e.g. TDM, static and dynamic priority based arbitration performed locally in routers, rate control along with round-robin schedul-

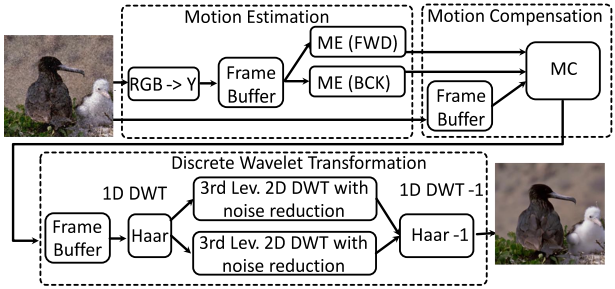
ing in routers etc.

Overview of the Benchmarks

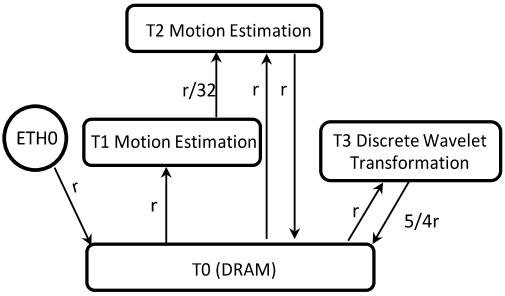
The benchmarks can be divided into two classes of traffic sources: i) synthetic traffic generators and ii) memory trace-based processor models. The former initiate transmissions according to a predefined synthetic pattern. This approach is frequently used in (on- chip) network research as it allows covering the corner cases, i.e. worst-case behavior of the system. Consequently, the conducted experiments consider synchronization scenarios with x senders performing a burst of y transmissions per activation. For each sender it is possible to set the activation period P, a small jitter J, as well as the length of the transmission L setting out how many packets are issued per activation. In order to imitate irregular traffic patterns, e.g. sporadic bursts, either the Bernoulli or Markov processes have been applied. Furthermore, the considered synthetic sources allow the simulation of streaming senders (e.g. video cameras, decoders etc.) which impose a high load while exhibiting predictable access patterns. Consequently, in the first part of the experimental evaluation two use-cases with streaming senders are used: real-time video noise reduction application [95] and MPEG-4 [19].

Figure 5.1(a) presents the structure of the algorithm used for the real-time video noise reduction [95]. Its workings can be divided into three phases: motion estimation (T1), motion compensation (T2) and discrete wavelet transformation (T3), and further decomposed into communicating tasks as presented in Figure 5.1(b). The data rate r is relative to the video data rate. In the conducted experiments, the data rate r = 318 MB/s has been used, which can be translated into 40 fps for a 1080p (1920x1080 pixels) resolution or 26 frames for a 2K (2048x1556 pixels) resolution. For evaluation, each task from the noise reduction application is modeled with a periodic traffic generator. Periods are adjusted to conduct 8kB long DMA transfers to maximize the benefit from DDR3-SDRAM [68]. A detailed evaluation of the memory effects will follow in the next sections.

The MPEG-4 video decoder [19] has been modeled using a similar methodology. Figure 5.2 presents the structure of the implemented algorithm with bandwidth requirements presented in MB/s (video resolution HD: 1920x1080 @ 30 Hz [48]). For evaluation purposes, three modules (with especially high communication requirements) have been identified: DRAM (the target of seven senders), SRAM2 (target of four senders) and SRAM1 (target of two senders). Transmissions to each memory module constitute an independent synchronization scenario which is mapped to an independent VC and protected with a RM. Similarly, each module of the MPEG-4 decoder is modeled with a traffic generator periodically conducting 8kB long DMA transfers to increase performance of the DDR3-SDRAM 2133N, cf. [68]



(a)



(b)

Figure 5.1: Usecase algorithm for film grain noise reduction [95] (a) and its average communication demands (b).

module.

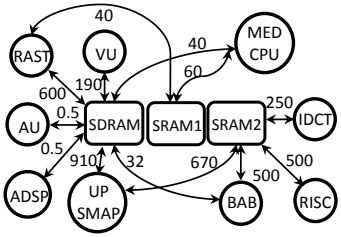


Figure 5.2: MPEG-4 average communication demands specified in MB/s, cf. [19].

Finally, some conducted experiments used traces of memory accesses from real

applications belonging to the CHSTONE benchmark suite [51]. An overview of the used applications is presented in Table 5.1. The traces of benchmarks are extracted using the Gem5 simulation platform [21] with an ARMv7-a core (1GHz frequency and a single cache level with 64kB and a block size of 64 bytes). The compilation process was done with the standard GCC compiler (v4.7.3), i.e., it has not been optimized with respect to the QoS mechanisms.

Furthermore, benchmarks can be divided into computation and memory intensive groups, based on the characteristics of the memory traffic, the frequency of accesses and the duration of the on-core execution. The computation intensive applications conduct between 1500 and 4000 transmissions every millisecond (e.g. *mips*, *motion*, *gsm*) whereas the memory intensive applications conduct between 10000 and 16000 transmissions every millisecond (e.g. *adpcm*, *dfdiv*, *aes*, *dfmul*).

Program	Design Description	Source
DFADD	Double-precision floating-point addition	SoftFloat [60]
DFMUL	Double-precision floating-point multiplication	SoftFloat [60]
DFDIV	Double-precision floating-point division	SoftFloat [60]
DFSIN	Sine function for double-precision floating-point numbers	CHStone group, SoftFloat [60]
MIPS	Simplified MIPS processor	CHStone group
ADPCM	Adaptive differential pulse code modulation decoder and encoder	SNU [87]
GSM	Linear predictive coding analysis of global system for mobile communications	MediaBench [90]
JPEG	JPEG image decompression	The Portable Video Research Group [64], CHStone group
MOTION	Motion vector decoding of the MPEG-2	MediaBench [90]
AES	Advanced encryption standard	AILab [9]
BLOWFISH	Data encryption standard	MiBench [58]
SHA	Secure hash algorithm	MiBench [58]

Table 5.1: List of benchmark programs belonging to the CHStone suite [51].

Configuration of the Baseline NoC

The used NoC architecture complies with the baseline architecture, described in Chapter 3, if not stated differently. It supports a 2D mesh topology, cf. Figure 5.3, wormhole switching, source routing (XY as a baseline), two stage priority-based arbitration, 4-cycle router pipeline, 1-cycle link traversal, virtual channels (VCs), non-blocking routers, a buffer depth of 5 flits, a data packet size of 4 flits, and an interconnect frequency equal to 500MHz. The arbitration between streams with the same priority is solved using the round-robin scheduler. Control messages (belonging to the synchronization protocol) are propagated using the VC with the highest priority.

Additionally, in the considered architecture nodes can be heterogeneous and constructed of processing cores (from T1 to T12) as well as peripherals, e.g., I/O interfaces, Ethernet or DRAM controllers. Moreover, as in many commercially available NoC-based architectures, certain peripherals (e.g. memory hardware units) have a fix position in the system which cannot be changed by mapping (e.g. edge of the chip due to the physical size of the controller). Therefore, certain critical communication paths are also fixed (i.e., established by the static routing algorithm), e.g., communication from the Ethernet controller to the DRAM memory.

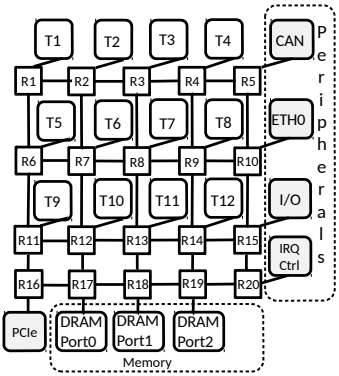


Figure 5.3: Exemplary NoC-based MPSoC setup used in the experiments.

5.2 Worst-Case Guarantees

This chapter presents results from the formal verification of setups with the different versions of the synchronization protocol (cf. Chapter 3). In consecutive runs, the properties of the baseline NoC enhanced with the control layer are compared to the state-of-the-art solutions described in Chapter 2. Consequently, Section 5.2.1 considers the time-driven NoC designs whereas Section 5.2.2 considers priority based schedulers in routers. Finally, Section 5.2.3 evaluates the worst-case communication overhead introduced by the synchronization protocol. The proposed analysis can be directly extended to account for the overhead resulting from the SW/HW stack using the CPA principles [61].

5.2.1 Time-Driven Scheduling

In consecutive series of experiments, synchronized senders are activated periodically with a period $P = x \cdot C^+$ and a small jitter J equal to 10% of a period. This permits varying the system's load L set out as the total number of transmissions from the synchronized applications per period P i.e. $L = \sum_x (y_x \cdot C^+)/P$. Moreover, all synchronized transmissions are of equal length, i.e., they have the same C^+ equal to 16kB. Figure 5.4 summarizes the results from the first series of experiments, where a setup with four ($x = 4$) applications ($A1$-$A4$) is considered. In each run, the lengths of bursts are adjusted to generate different loads L on the network (L is equal to 15%, 65% and 90% of P). For each application, the worst-case latencies obtained per burst are calculated.

It is visible that synchronization with the RM always results in better guarantees than TDM (even when short slots are used) despite the additional communication protocol. The overhead considers the protocol-based synchronization without the reaction time of clients and the RM. This overhead increases with the load but remains low compared to the transmission time (4,1% of C^+ for L=90%).

In the next series of experiments the variable load has been tested. Figure 5.5 illustrates results from the previously considered setup, i.e. transmission latency and protocol overhead in the worst case, while changing the number of synchronized applications ($x = [2..16]$) and interfering load ($y_x = [2..16]$). The results concern an application periodically transferring a burst of 16 transmissions. Each single transmission from all synchronized senders has an equal length, i.e., they have the same C^+ equal to 8kB.

The worst-case latencies, depicted in Figure 5.5(a), for both TDM and the RM, increase along with the size of the synchronization scenario. However, they remain constant for TDM and independent of the system's load. Therefore, the approach

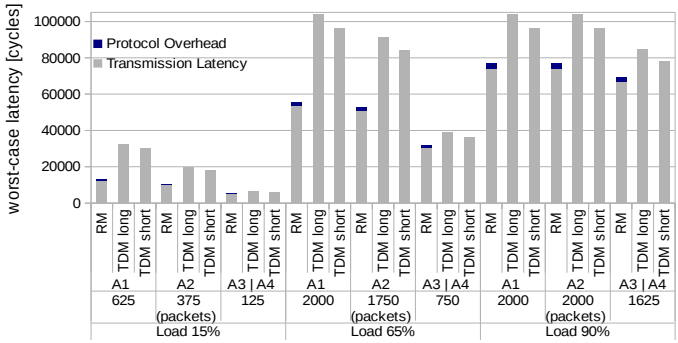
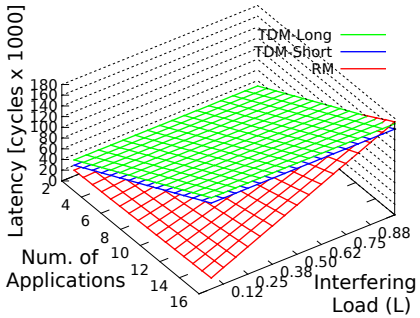


Figure 5.4: Analytical comparison of worst-case latency guarantees for applications ($A1$-$A4$) generating different NoC load. $A3$ and $A4$ have the same settings, see [81]. The overhead considers the communication protocol i.e. latency of synchronized messages.

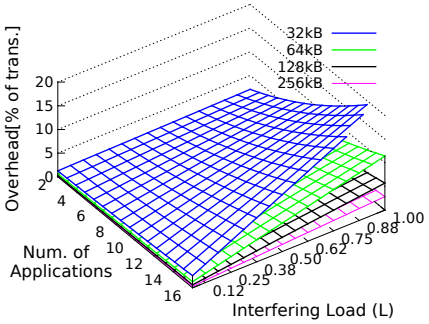
based on the control layer significantly outperforms TDM (up to 80%), due to the applied work-conserving scheduling. Note that TDM with short slots performs better than TDM with long slots as it is less sensitive to the jitter.

The protocol overhead, depicted in Figure 5.5(b) is presented as a percentage of the transmission length. It increases proportionally to the number of synchronized senders. This effect can be mitigated by implementing multiple RMs in the same NoC. Moreover, the protocol overhead depends directly on the frequency and number of synchronized transmissions, i.e., system load. Finally, this overhead decreases, as an absolute ratio, with an increasing length of transmissions as the protocol overhead is constant with respect to the transmission length.

The next series of experiments compares the *service guarantees* provided by the RM and TDM for an application performing a *burst* of 16 consecutive transmissions, see Figure 5.6. Similarly, to the previous experiment, in consecutive runs the number of synchronized applications x increases ($x \in [2..16]$). Each application performs a burst of transmissions per activation. Applications are activated periodically with a period $P = 16 \cdot x \cdot C^+$. Consequently, the load L defined as the number of transmissions $y = [2..16]$ from interfering applications varies, per a single burst activation, i.e., $L = \sum (y \cdot C^+)/P$. Moreover, transmissions are not perfectly synchronized with the TDM schedule (jitter equal to 5% of C^+). Figure 5.6 presents the results. In the system with TDM, the worst-case service depends directly on the number of synchronized applications and does not vary with the load.



(a)



(b)

Figure 5.5: Worst case guarantees for a burst of 16 transmissions with jitter =10%P (a) Transmission latency (b) Protocol overhead resulting from RM. Results are based on [81] and [82].

This results in a low scalability and poor support for dynamics. For the RM-based admission control, it is possible to achieve better service guarantees when the system is not fully loaded ($L < 100\%$). This is due to the introduced work-conserving arbiter which schedules transmissions as soon as they arrive, in contrast to the static overhead of TDM. Consequently, if the NoC's load is low, the service guarantees for the sender (*RM-Low*) remains continuously on the same low-value regardless of the number of synchronized senders. Note that the service guarantees for other load levels span proportionally - green series (*RM-High*) in the Figure 5.6. For a fully loaded system, the overhead resulting from the synchronization protocol is also acceptable ($\simeq$4% compared to TDM). Such differentiation is not possible in

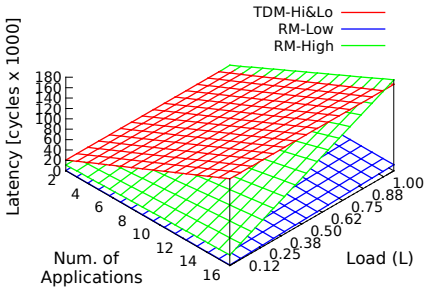


Figure 5.6: Worst-case service for sixteen consecutive transmissions in a system with TDM-based arbitration and compared to the RM-based architecture, based on [83].

case of TDM, as TDM offers a non work-conserving arbitration scheme according to which guarantees depend on the number of synchronized applications and not on the load which is induced by them (cf. [59]).

Memory Effects

The next series of experiments extends the evaluation to account for memory effects caused by the proposed arbitration. For evaluation purposes, the following DDR3 memories have been considered: 800E, 1066G,1066E, 1333H, 1333J, 1600K, 1866M and 2133N. Firstly, the effects of spatial locality on the response time of a memory module have been evaluated, assuming the simple SDRAM controller presented in Sec. 3.8 of Chapter 3. In the consecutive series of experiments, a benchmark application tries to write between 8 to 1024 bytes of data into the memory in systems with (solid lines) and without (dashed lines) support for locality of transfers, cf. Figure 3.28 from Chapter 3. In the context of this experiment, to support locality means to ensure that the entire chunk of data arrives in one piece at the SDRAM controller, which would be the case in a system with large TDM slots or a RM. To not support locality means that several independent small requests are performed in order to transfer the entire data chunk of a request, which is the case in a system with small TDM slots.

Figure 5.7 summarizes the obtained results. For all tested SDRAM devices, whenever locality is enforced, the latency of a request can be significantly decreased. The performance gain is proportional to the length of transmissions and scales with the frequency of the memory module, see the zoomed section of the figure. The faster the frequency of the SDRAM device, the larger the latency measured in clock cycles. This is because SDRAM timing constraints vary little between differ-

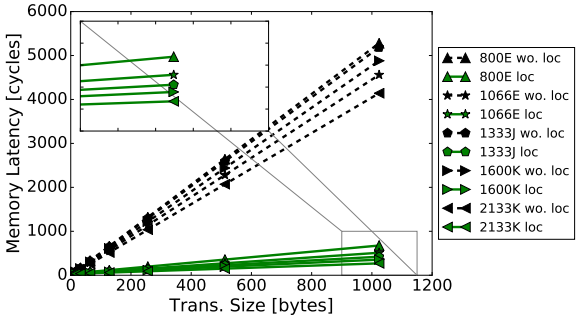


Figure 5.7: Effect of memory locality on the total transmission latencies for MPEG-4 module using TDM and RMs, based on [82]. The calculations assumed a SDRAM device with an interface width of 8 bits. Notice that the latency is measured in data bus clock cycles of the SDRAM devices.

ent speed bins when measured in nanoseconds. Consequently, devices that have faster clock frequencies need more clock cycles to satisfy the constraints.

The former experiment shows that increasing the transmission granularity results in decreasing memory overhead in a NoC-based MPSoC. However, for TDM-based arbitration, this action simultaneously increases the worst-case NoC latencies. Recall that, whenever the senders in a system exhibit dynamics in their behavior, even with a small jitter, transmissions are blocked and their execution is delayed for the duration of a whole TDM cycle.

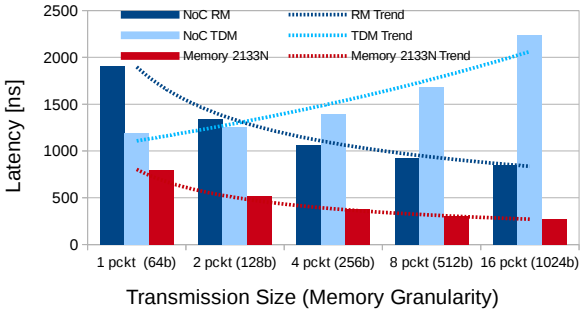


Figure 5.8: Effect of memory locality on the latencies of resource accesses with different granularity, based on [82].

Therefore, longer transmissions increase this penalty. Figure 5.8 presents the worst-case latencies in a NoC-based MPSoC for an application synchronized with 10 other senders and writing 1024 bytes to the SDRAM memory. The experiment is conducted under the assumption that each packet is capable of delivering 64 bytes of payload. In the consecutive runs, the granularity of access has been increased conducting the synchronization for appropriately 1, 2, 4, 8 and 16 packets assuming a small dynamic behavior in the activation of a sender (round 5%). Consequently, the load from other synchronized senders is equal to 70% of the network capacity. In case of TDM, the NoC latency increases proportionally to the length of the slot, i.e., number of packets. Consequently, decreasing the memory latency increases the NoC response time. In contrast, when the RM is used, the latency decreases simultaneously with increasing granularity of the transfer, i.e., it is possible to simultaneously improve network and memory latency. However, for a short transmission (1 or 2 packets), the RM's protocol overhead is a major performance bottleneck (recall that three ctrl. messages per transmission are required). In these scenarios, TDM is managing to guarantee lower transmission latencies. Yet, as soon as the granularity of a single transfer increases, the number of necessary synchronizations depletes. This results in a significant improvement for the 1024 bytes long transmission (almost 70%) and confirms previous findings.

Finally, the last series of experiments demonstrates that preserving the locality of large transfers can have a beneficial impact on the worst-case latency of SDRAM requests. In the considered setup a SDRAM controller with a single port is connected to the NoC. For the RM-managed NoC, the simple SDRAM controller is used and all requests are treated equally (no QoS support, performance optimized cf. Sec. 3.8 of Chapter 3). In case of the TDM-managed NoC, a non-trivial predictable SDRAM controller is considered to employ static bank interleaving and closed-page policy. This controller will be referred as Dedicated Close Page-Controllers (DCPC) in the following. The operation of the considered DCPC is controlled by two parameters: Bank Interleaving (BI) and Burst Count (BC). BI determines how many banks a single request accesses while BC determines how many read or write commands are executed per bank.

The evaluation consists in computing the worst-case latency of a 256-bytes long data transfer. For the TDM-based NoC, 64-bytes long slots are considered, i.e. the 256-bytes transfer will demand 4 independent SDRAM requests. For the RM-managed NoC (which enforces the locality of a single large transfer), the 256-bytes long transfer is served in one chunk.

The results are presented in Figure 5.9(a) and Figure 5.9(b). The first figure depicts the results for a scenario in which a single SDRAM device with an 8-bit data bus is considered. Notice that, for SDRAM devices with slow operating frequencies

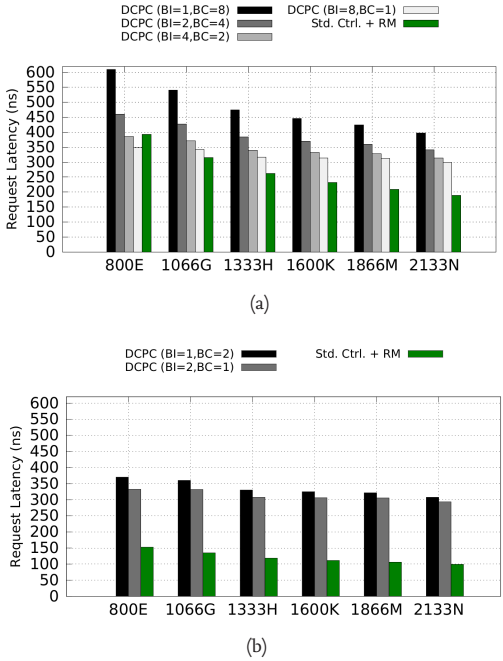


Figure 5.9: Worst-case latency for a 256-bytes request on DDR3 devices (a) with 8-bit wide interfaces and (b) with 32-bit wide interfaces connected to a NoC with and without RM (locality protection), based on [82]. It is visible that RM (green series) consistently reaches shorter formally proven latencies.

(DDR3-800E), the DCPC provides better latency bounds than the combination of a standard controller and the RM. This is because, for slow devices, the penalty to close and open SDRAM rows is smaller (in terms of data bus clock cycles). This situation changes for devices with higher operating frequencies because the overhead becomes larger.

The second figure depicts the results for a scenario in which a SDRAM module with a 32-bit data bus is considered. As the data bus is larger, the possibility to perform interleaving is reduced (because each SDRAM CAS command transfers 4 times more data in comparison with the previous scenario.) Hence, there is no efficient way to mitigate the overhead to close and open SDRAM rows. Therefore, exploiting the locality of large transfers has a solid advantage.

5.2.2 Priority-Based Arbitration

This section presents results comparing the worst-case guarantees provided by RMs against the system utilizing prioritization of VCs (*VC-prio*) carried out locally in routers as proposed in [31], [24], see Figure 5.10. The same experimental settings as before are assumed with a variable number of synchronized applications. Each application has a unique priority and periodically conducts burst-accesses to the NoC. All transmissions belonging to the same burst are of equal length (125 packets) and have the same maximum network latency C^+. An activation period of application equal to $P = 16 \cdot x \cdot C^+$ and a small jitter equal to 5% of C^+ is assumed. In the experiments, the number x of synchronized applications ($x = [2..16]$) is adjusted. Each sender has a unique priority and conducts burst-accesses to the NoC with l transmissions per activation. We assign priorities to applications using the rate monotonic priority assignment.

The considered mechanism for prioritization of VCs offers preemption at the flit-level. At any time a packet with a higher priority gets the privilege to proceed in the router and access the output link. Figure 5.10 presents the results for the synchronized applications with the lowest priority. Latencies are depicted with bar-charts and refer to the left Y-axis. Both approaches provide work-conserving scheduling of the network traffic, therefore results are comparable. It is visible that prioritization of VCs offers better worst-case guarantees as it does not require a synchronization protocol like the RM. However, the mechanism requires that the number of VCs is equal to the number of criticality levels in the system. At the same time, the RM requires only two VCs (including one for control messages) for all applications. The hardware overhead, defined as the number of VCs, is denoted with dashed (*VC-prio-Lo*) and solid (*RMs-Lo*) lines and refers to the Y-axis on the right. At the cost of roughly 13% of additional latency overhead, the RM significantly reduces the hardware overhead proportional to the number of criticality level in the system. In the example, this overhead is divided by 8 (16 VCs in VC-prio vs. 2 VC in RM).

In the second series of experiments, the RM-based approach is compared against the mechanism based on prioritization of VCs (flit preemptive scheduling performed locally in routers) while considering memory latency. Although both solutions offer work-conserving scheduling, the latter one does not preserve the locality of synchronized transmissions (cf.Sec. 3.8 of Chapter 3). For evaluation, the DRAM main memory module was considered with worst-case response latencies modeled after the specifications of DDR3-SDRAM 2133N. In these experiments, the DRAM synchronization scenario from the MPEG-4 use-case has been considered with 7 synchronized senders. Similarly to the previous experiments, the priori-

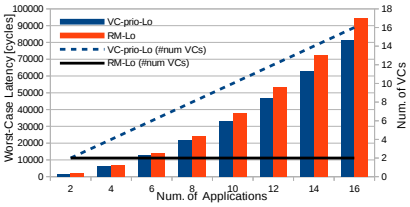


Figure 5.10: Worst-case response times in a system with prioritized VCs and RMs, based on [83].

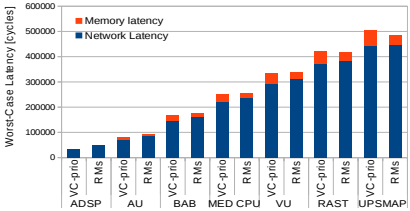


Figure 5.11: Effect of memory locality on average latencies for prioritized VCs and RMs, based on [83].

ties are increasing along with decreasing bandwidth requirements. Therefore, in scenarios utilizing the prioritization of VCs one has to assign seven VCs, whereas the RM requires only two VCs (one for control messages and one for data transmissions). In the case of the RM preemption at the granularity of $8KB$ has been considered (similar to main memory paging), i.e. the preemption may happen only after the transfer completion of 8KB chunk of lower-priority transmission. This allows applying the open-bank policy in the selected memory module, as described in [153]. The solution based on prioritization of VCs does not offer this possibility as the preemption happens, locally in routers, on the flit level.

Figure 5.11 presents the worst-case latencies obtained with the proposed analysis method. It is visible that although prioritization of VCs results in lower network latency, it increases the latency of memory operations. Consequently, the RM achieves better overall results, in particular when the most of the streams are of lower priorities.

5.2.3 Worst-Case Temporal Overhead

In order to assess the scalability of the approach, a series of experiments has been conducted to measure the worst-case temporal overhead by varying the number of synchronized applications and the system load. Consequently, the number of applications (and induced traffic) using the same RM has been increased from 2 to 16 in consecutive runs. Figure 5.12 summarizes the results as a percentage of the transmission latency which is understood as the time necessary to send 8kB of data using 125 packets in a NoC without interference. The overhead increases proportionally to the number of synchronized applications and their activity, i.e., load L. Note that this is directly related to the frequency of transmissions. If the system is composed of many applications that are rarely sending data, then the

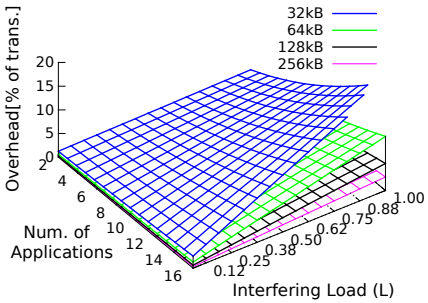


Figure 5.12: Temporal overhead resulting from RM, based on [83].

overhead remains on a low level. Moreover, in the considered safety critical systems, the behavior of safety-critical is well specified and tested, and the frequency of transmissions with hard real-time requirements is limited (e.g. periodic in ms).

It is also visible that the overhead decreases, as an absolute ratio, with an increasing length of transmissions, i.e., the overhead is constant with respect to the transmission length. DMA engines imposing large granularity of transfers combined with scratchpad memories are mostly used for safety requirements as they are predictable when compared to caches.

The temporal overhead can be mitigated by implementing multiple RMs in the same NoC (for example by providing different RMs for different memory modules or regions of the NoC). Consequently, large synchronization scenarios can be decomposed into smaller ones by mapping transmissions to different VCs supervised by independent RMs. Additionally, for senders conducting short transmissions (e.g. cached based applications) the synchronization can be performed per whole on-core task activation rather than for single NoC accesses, cf. Chapter 3.

5.3 Performance Measured by Simulation

In this section, the performance of the proposed mechanism is evaluated using simulation. The arbitration is implemented using the OMNeT++ framework for network simulation and the HNOCS library [15]. A detailed description of the framework is presented in Chapter 4 Section 4.1.2.

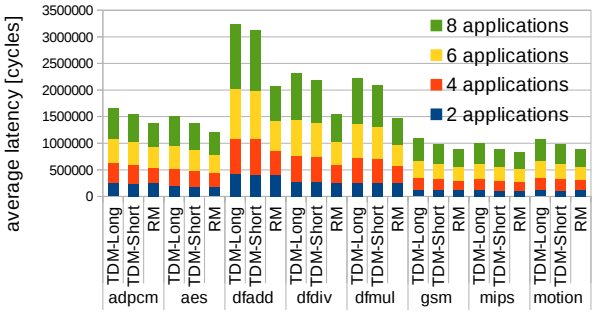


Figure 5.13: Latencies of CHSTONE benchmark with TDM and RMs, based on [81].

5.3.1 Performance of Time Driven Scheduling

Firstly, the evaluation is carried out through average performance measurements using traces from the CHSTONE benchmark [51]. In consecutive runs, different synchronization scenarios have been simulated by varying the number of senders. For each synchronization scenario (selected number of applications) all possible mappings have been simulated assuming a constant placement of the RM and one sender per node. Figure 5.13 presents the average latencies for different sizes of synchronization scenarios. In this case, it is visible that the RM significantly outperformed other solutions. However, when compared to TDM for small synchronization scenarios, e.g. 2 senders, the difference is rather low (around 8%). This has two reasons: a short duration of TDM-cycles and a relative high RM overhead (three control messages per transmission). For larger scenarios, the solution based on RMs is up to 60% better than TDM.

Note that the activation patterns of senders are not necessarily synchronized with respect to the TDM-cycle. Tailoring the TDM schedule in order to optimize such systems, which are not fully loaded, requires dedicated solutions and introduces additional hardware overhead, e.g. SurfNoC [150], PhaseNoC [116]. The RM conducts arbitration efficiently without additional effort.

Secondly, the average performance of RM-based arbitration has been evaluated in the use-case using the MPEG-4 video decoder application [19]. This comparison is relevant for all cases in which a synchronized application not only requires the worst-case guarantees but also profits from a faster execution.

In the MPEG-4 usecase, requests to each memory module constitute synchronization scenarios. Consequently, different scenarios are mapped to independent VCs and protected with a dedicated RM running on a different node, see Fig-

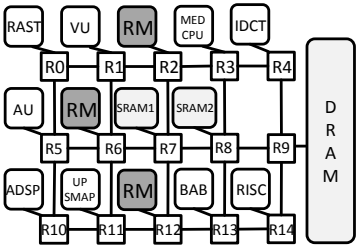


Figure 5.14: Exeprimental setup used for the evaluation of locality effects on the total transmission latencies for MPEG-4 module, based on [82].

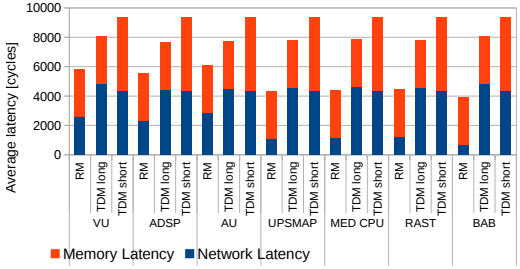


Figure 5.15: Effect of memory locality on the total transmission latencies for MPEG-4 module using TDM and RMs, based on [82].

ure 5.14. The following concentrates on the DRAM scenario. Each module of the MPEG-4 decoder is modeled with a traffic generator conducting 8 KB long DMA transfers. This is because, in commercial SDRAM modules with 64 bit data buses, the row buffer size is 8 KB.

Transmissions are performed periodically and periods are calculated based solely on the required bandwidth including some release jitter (J=10% of P). Figure 5.15 presents the achieved average latencies of a single transmission in the simulated system with TDM- and RM-based arbitration. The transmissions are 125 packets long which corresponds to 8kB of data payload in the considered baseline NoC. The depicted values include both network and memory latencies to assess the effect of memory locality on the described mechanisms. Latencies of the SDRAM memory are modeled after the specifications of DDR3-SDRAM 2133N [68]. As previously explained, TDM with short slots performs better than TDM with long slots in the network. However, the memory latency for TDM with long slots is lower than for TDM with short slots since the long slots allow maintaining the locality of memory accesses to the SDRAM. Indeed, SDRAMs have an internal level of caching, which in standard DDR3 modules amounts to 8kB. Consequently, *contiguous and aligned* 8kB long transfers fully benefit from the caching.

The RM also maintains the locality of memory accesses, since applications are granted access to the NoC for the entire transmission. Overall, the RM performs better than TDM for memory traffic traversing the NoC and accessing main SDRAM memory. Similarly, to the worst case, the RM offers the lowest average latencies.

5.3.2 Work-Conserving Arbitration in NoC

As discussed in Section 5.2.2, the control layer permits priority based arbitration within the same virtual channel in a NoC. Consequently, when compared to architectures offering QoS based on prioritization of VCs done locally in router, it introduces additional overhead resulting from synchronization protocol. However, at the same time, the control layer decreases the number of required buffer queues (i.e., the number of priorities is independent of the available hardware resources) as well as increases the performance of state based peripherals e.g. locality of accesses to DDR-SDRAM memories.

Results from the conducted experiments confirm that the proposed resource management scheme can further improve the performance of running applications. The first evaluation is performed through a comparison of the RM introducing a slack-based global and a dynamic prioritization of data streams with SPP-based solutions. The former schedules BE and soft-real time (SRT) senders whenever the hard real-time (HRT) senders can be safely postponed. This is possible whenever a slack is available, which is the temporal budget between the worst-case response time of a HRT application and its deadline, cf. Section 3.6.2 from Chapter 3. For the SPP arbitration carried out locally in routers, SRTs and BE streams are always statically assigned the lowest priority. The slack budgets for

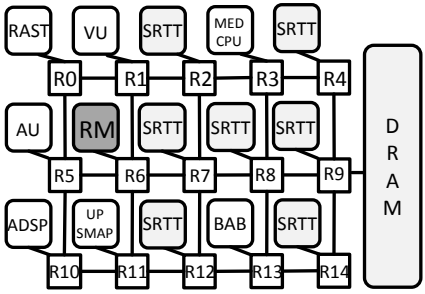


Figure 5.16: Experimental setup used for evaluation of the dynamic priority based scheduler in the RM, based on [75].

HRT senders, used in experimental runs, are computed offline with analysis from Section 4.1.1 (implemented in the pyCPA framework [42]). Figure 5.16 shows used manual yet sparse mapping for the DRAM synchronization scenario of MPEG-4 use-case. HRTs as well as SRTs are mapped to the same VC and synchronized with the same RM.

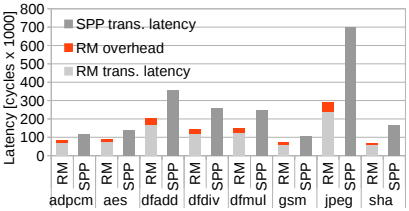


Figure 5.17: Performance of CHSTONE benchmarks (SRTTs) synchronized with MPEG-4, based on [75].

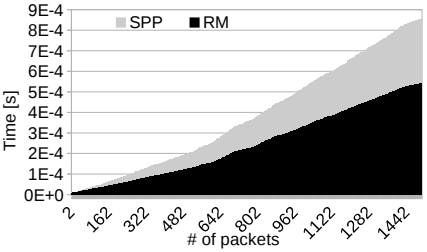


Figure 5.18: Trace with completed transmissions from the SRTT *dfmul* benchmark, based on [75].

The evaluation begins with an assessment of the performance gain for SRT senders achieved through the proposed mechanism. Figure 5.17 presents latencies of CHSTONE benchmarks (SRTs) when synchronized with tasks from MPEG-4 (HRTs) and belonging to the DRAM's synchronization scenario. Results concern both the SPP arbitration and the RM using slack-based scheduling. One may observe that benchmarks synchronized with the RM finish faster compared to the SPP despite the additional overhead resulting from the synchronization protocol. The improvement varies among applications to reach a speedup value of nearly 60% for jpeg application. This depends on the duration and the frequency of blocking in the NoC resulting from the access patterns of soft and hard real-time applications. The speedup is also proportional to the number of memory accesses, e.g. *jpeg* exposes the highest number of transfers and thereby highly benefits from the improvement in network latency. Finally, it is visible that the protocol overhead remains low compared to the transmission time (from 5 to 14 %).

For a detailed explanation, Figure 5.18 presents the evolution over time of the number of completed transmissions from the *dfmul* benchmark. It is visible that the application progresses significantly faster when synchronized with the RM. The slack-based resource allocation allows SRT to fetch data and start *early* processing, therefore accelerating the benchmark's execution when compared to SPP.

As previously explained, the slack budget available to SRTs varies at runtime. Figure 5.19 presents the evolution of the available slack over a time window of 200 ms in the considered DRAM scenario. Rising edges in the figure correspond to the provisioning of the slack budget (between S_{min} and S_{max}) when a new HRTT is activated, and falling edges to the consumption of the slack budget when a SRT is granted access to the NoC. It is visible that the value of the available slack varies between the initial slack budgets (S_{min}/S_{max}) and ψ. Consequently, as the slack never

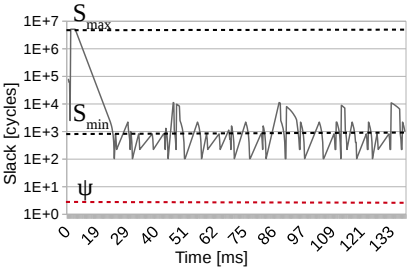


Figure 5.19: Slack budget monitored by the DRAM's RM, based on [75]. (Available slack $\geq$ SRTTs requirements).

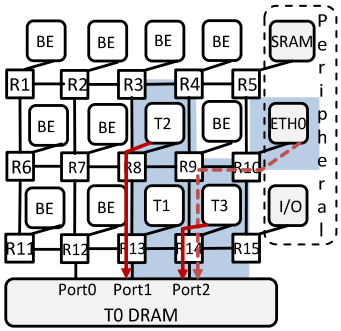


Figure 5.20: Experimental setup with mapping of HRT tasks performing video denoising. Network demands are presented in MB/s for 1080p and 40 fps, based on [77].

reaches ψ, SRTs can be safely accelerated by postponing the execution of HRTs without causing a deadline miss. Moreover, it is visible that the available slack stays on a relatively high level (compared to ψ). This means that the provided slack budget is larger than the SRTs' requirements. Therefore, the average performance of HRTs has not decreased significantly.

Adaptive Load Distribution

As discussed in Section 3.6.5, the introduced control layer also offers the possibility to detour traffic instead of blocking it by activation of higher priority senders. Consequently, BE senders may use multiple paths from the source to the destination, even if they overlap with links used by critical traffic. Nonetheless, whenever safety HRT senders are active, these links are treated as the ones with the highest contention and BE traffic is detoured to an alternative path - instead of blocking it permanently as a state-of-the-art solution would do. As will be shown, such arbitration results in the significantly improved average latencies of BE senders. Therefore, the RM is gradually enhancing BE performance whenever the NoC contention is high.

In this series of experiments, HRT senders are modeled after the real-time multimedia application performing video noise reduction [95], see Figure 5.20. The HRT traffic is distributed between the DRAM ports due to the conflicting temporal (bandwidth) requirements: T2 into the port1, ETH & T3 into the port2. BE senders are modeled after applications from the CHSTONE benchmark. A comparison is performed for three mechanisms: proposed adaptive load distribution

(ALD) and other common solutions for mixed-critical NoCs: spatial isolation (SIS) and temporal isolation (TIS) done by static priorities.

SIS enforces all BE senders to use the remaining *PORT* 0. For ALD and TIS the load from BE senders is distributed between all the available ports (three BEs per port using the shortest path and mapping as in [52]). Consequently, for ALD and TIS, memory ports constitute three hot-modules and interfering senders using each of them can be synchronized independently. ALD enforces one detoured path per BE to *PORT* 0 - assured by XY-routing and non-blocking routers. For all HRTs, the overhead against the slack budget was confirmed with the formal analysis done in the PyCPA framework [42] (for ALD overhead around $\sim$ 5% of transmission time and between $\sim$ 0.1 to 0.2 % of period).

Figure 5.21 presents the achieved results - these values consider both the core execution and the network communication (interference). It is visible that SIS offers safety at the cost of drastically decreased BE performance i.e. the longest transmission latencies due to the strict separation resulting in the high interference in *PORT* 0 and the increased average path's length. For TIS, the distribution of the BE traffic between the ports decreased the average transfer latencies and improved the average BE performance. However, due to the temporal synchronization the improvement is proportional to the HRT load (i.e. the duration of blocking) therefore the improvement is better for *PORT* 1 (one HRT sender) when compared to *PORT* 2 (two HRT senders). Finally, ALD allowed further performance improvement for traffic directed to *PORT* 1 and *PORT* 2 at the cost of a minimal slowdown of applications accessing *PORT* 0. If HRTs are active, the traffic from BE senders is redirected (detoured) to the *PORT* 0 causing an interference with benchmarks which were running alone in the previous run (TIS). However, as detouring happens only during conflicts with HRT, on average (all benchmarks) 37% improvement is reported when compared to SIS and 20% of an improvement when compared to TIS. Note, that the difference between TIS and ALD is higher for *PORT* 2 (higher HRT load) than for *PORT* 1 (lower HRT load) as only the blocked traffic can profit from the detouring (ALD), in contrast to TIS.

For a more detailed explanation, Figure 5.22 presents the histogram of latencies from the *adpcm* benchmark running on the node connected to the R9. If the NoC is protected by ALD most packets can progress using the optimal path (short latencies) and only a few are detoured causing the gradual decrease of the performance. In case of SIC, all transmissions must take a longer, non-optimal, path and experience a higher interference from other BE senders - thus with significantly higher average latencies. Finally, in case of TIS although some transmissions may progress quickly (similarly to ALD), others are blocked by one or two consecutive HRT transmissions (adequately around 1.5x and 2.5x depending on activities of

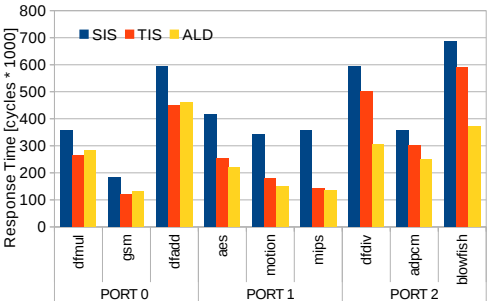


Figure 5.21: Performance of CHSTONE benchmarks (BEs) in the usecase with ALD, based on [77].

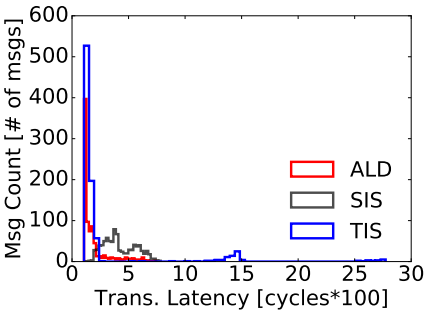


Figure 5.22: Histogram of transmission latencies for *adpcm* benchmark in the ALD use-case, based on [77].

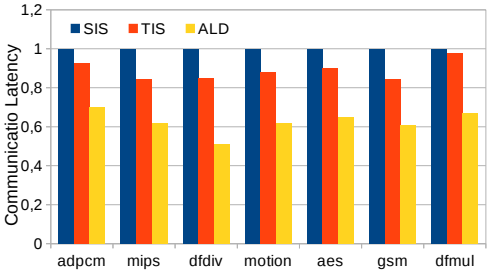


Figure 5.23: Improvement for different BE benchmarks in the ALD use-case normalized to SIS, based on [77].

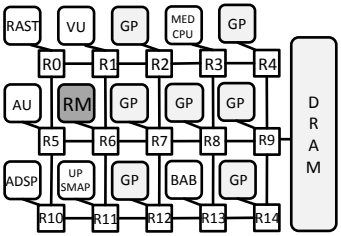


Figure 5.24: Effect of memory locality on the total transmission latencies for MPEG-4 module using TDM and RMs, based on [77].

*ETH*0 and T3) jeopardizing the average performance.

The performance improvement depends heavily on the characteristics of memory access from the benchmarks. Figure 5.23 presents the improvement (normalized w.r.t to SIS) for different benchmarks running on the node connected to the R9 (average from all possible benchmarks mappings). The improvement is significant and varies among benchmarks starting from 25 % for *adpcm* to reach a speedup value of nearly 55 % for *dfdiv*. The speedup is proportional to the frequency and number of memory accesses as well as the activity of other senders, i.e. a memory intensive application conducting more transfers experiences a higher improvement in the response time than a computation intensive task.

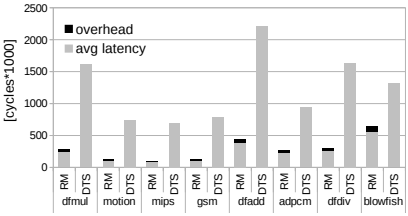


Figure 5.25: Mapping of the MPEG-4 with interfering traffic from general purpose (GP) applications in the DTS use-case, based on [78].

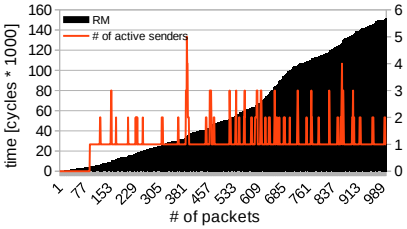


Figure 5.26: Trace from dfmul benchmark with the number active senders showing the number of packet arrivals in the DTS use-case, based on [78].

Dynamic Rate Control

This section presents results proving that the control-layer can efficiently control rates at which running applications access the NoC. At each source node, the monitor regulates the rate with which the source can inject traffic in the NoC. The major drawback of this solution is that the rates are adjusted statically according to the worst-case scenario, i.e., assuming that all senders are running simultaneously with maximum possible transmission arrival rates. Therefore, this approach is not work conserving and sacrifices the interconnect utilization whenever senders expose dynamics in the execution time, release jitter or communication volume and the system is not highly loaded. In contrast, the control layer offers a solution according to which the regulation is performed dynamically (at runtime) according to the system load i.e. the number of simultaneously active applications. Consequently, the introduced approach results in a higher performance and tighter guarantees while simultaneously decreasing hardware (up to 60 %) and temporal overhead (up to 80%) when compared to existing solutions.

The evaluation starts with an assessment of the performance gain achieved through the proposed mechanism. Figure 5.25 presents the average response times of CHSTONE benchmarks running in the considered scenario, with mapping presented in Figure 5.24. Results include the distributed traffic shaping (DTS) arbitration in NI (rates adjusted and fixed according to the worst-case requirements in NIs and not in routers) as well as the RM approach (rates decreasing uniformly with the increasing number of active senders). One may observe that benchmarks synchronized with the RM finish faster despite the additional overhead resulting from the synchronization protocol. The improvement is significant and varies among

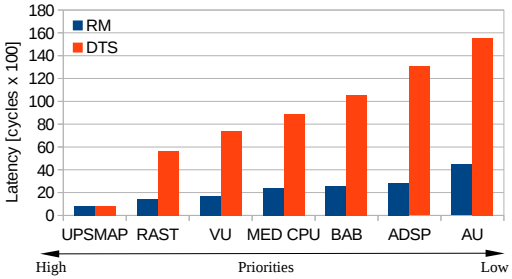


Figure 5.27: Average latencies for DTS and RM in the system requiring non-symmetric guarantees, based on [78].

benchmarks starting from 50 % for *blowfish* to reach a speedup value of nearly 85% for *mips*. This depends on the frequency and access patterns of initiated transmissions as well as on the activity of other senders. The speedup is also proportional to the number of memory accesses, i.e. an application conducting more transfers experiences a higher improvement in the response time due to the dynamic rate RM arbitration. It is also visible that the protocol overhead remains low compared to the transmission time ($\sim$ 15%).

For a detailed explanation, Figure 5.26 presents the evolution of the first 1000 accesses of the *dfmul* benchmark in the system using the RM along with the variation of the number of active senders (right Y-axis). It is visible that the benchmark's speed depends on the present system mode, i.e. packets latency varies according to the number of senders running in parallel. In the simulation, one may observe a maximum number of 5 active applications running in parallel with *dfmul*, for a total number of 8 synchronized applications. Moreover, during 60% of the time there is only one application running together with the considered benchmark. Consequently, the RM may significantly increase the transmission rate and provide high performance improvement at runtime. This performance gain results from the moderate load of the system during runtime.

The next comparisons concern the average latencies of transmissions synchronized with the RM against the system utilizing DTS when non-symmetric guarantees are required. The same experimental setting as previously is applied this time in a priority based system, where MPEG-4 tasks communicating with the DRAM memory are assigned priorities: tasks with higher frequencies are assigned a higher priority according to a rate monotonic scheme.

Recall, that each task conducts periodically burst-accesses (of equal length) to

the NoC. DTS used to enforce non-symmetric guarantees applies a *static* minimum distance function between two transmissions according to the priority, i.e. a shorter distance (high rate) in case of high priority transmissions. In case of the RM, the rates of tasks can be increased whenever it is possible, i.e other tasks are not active. The rate decreases with the increase in the number of active senders, however it cannot be lower than the rate which allows all active applications to meet their timing constraints.

Figure 5.27 presents the results for MPEG-4 decoder tasks. It is visible that when DTS is used, the average latencies increase along with the decreasing priorities of senders which get a low rate according to their priority level. Whenever the RM is applied, applications with lower priorities benefit significantly from the unused bandwidth by increasing their rate whenever it is possible (i.e when higher priority applications are not active), therefore achieving an increasing speed of up to 30% for the UPSMAP module. The transmissions with higher priorities achieve similar average latencies as for DTS.

5.4 Resource Overhead

As discussed, the proposed control layer introduces a resource synchronization scheme which can be applied to the broad spectrum of the underlying NoC architectures if they comply with some general requirements established in Chapter 3 e.g., predictable arbitration in routers, adjustable admission control in nodes, a predictable latency of control messages. In most setups, only small extensions of the underlying infrastructure are necessary, e.g., additional registers/interfaces in NIs, client logic. Moreover, the solution also offers fine-granular resource control, e.g., implantation entirely in hardware or HW-SW co-design for minimizing the overhead. This section presents the results of the initial evaluation of the resource overhead introduced by the mechanism. The initial designs followed as an extension of the baseline architecture are discussed in Section 3.1.

Hardware Implementation

The implementation of the control layer requires the introduction of a new RM unit and clients. The considered hardware implementation follows the design guidelines from Section 4.2.3. The RM is implemented in the form of a stand-alone processing node connected with one of the existing network modules. The clients are implemented as an extension of the NI in a system with a memory mapped NoC, where the NI introduces a transparent address translation using LUTs, see

Section 3.1. Moreover, the NI is already equipped with the mechanism for admission control, i.e., a rate-limiter which enforces a minimum temporal distance between the initiated transmissions.

The IDAMC platform [140] is selected as a basis for the implementation as it complies with all the requirements mentioned above. The IDAMC's routers' and NIs' architectures follow the generic design from [37] which is frequently applied by the research community. The deployment of the architecture is carried out on a Virtex-6-LX760 Xilinx FPGA board using *Xilinx ISE 14.6* with default optimization setting and no special optimizations for the *VHDL* implementation. The device utilization data were collected from the *Module Level Utilization Summary Report* produced by ISE.

Firstly, the results for several RM designs of increasing complexity are presented in Table 5.2. All considered approaches have been introduced in Chapter 3. The results consider solely the main RM logic, i.e., they do not include the NI functionality (e.g., packetization and depacketization). All RMs assume the same size of the stored tables with synchronization scenarios (64 entries). In the considered hardware implementation, the following RM schedulers have been used: static priority preemptive (RM-SPP) from Section 3.6.2, temporal isolation with round-robin (RM-RR) from Section 3.6.1, the adaptive load distribution (RM-ALD) from Section 3.6.4 (the RM instead of blocking low-level senders redirects their traffic) and finally, throttling using the dynamic traffic shapers (RM-DTS) from Section 3.6.5.

The overhead is compared with the size of an arbiter used to perform prioritization locally in router (fixed priority RTER-FP), see Section 3.1.1. The RTER-FP approach uses two different priorities in router and hence two virtual channels. In general, the RTER-FP scheme the number of available priorities to the number of VCs supported by the NoC.

The synthesis shows that the simple approaches (RM-RR, RM-SPP) introduce less than 5% overhead to the area of the NI module. The more complex approaches (RM-ALD, RM-DTS) introduce a slightly higher overhead (5.5%). The increase for RM-ALD and RM-DTS results from the additional information and control logic that is needed to estimate the path or rate limiter settings. However, all the approaches allow accounting for more priority levels that are available by the architecture. Hence, whenever more than two priorities supported by the scheduler in the router (RTER-FP) are necessary, the overhead for the RM-based approaches will not change. In contrast, the NoCs with classic prioritization done in routers will have to increase the size of routers by adding additional buffer space to prevent overflows or backpressure. The supplemental results for schedulers with even higher complexity are available in [146].

The initial evaluation of a client functionality is realized as an extension of the NI

Unit	RTER-FP	RM-RR	RM-SPP	RM-ALD	RM-DTS
#Regs	2581	2674	2702	2707	2715
#LUTs	4925	5093	5160	5160	5162

Table 5.2: Synthesis results of a RM on a Virtex-6 LX760 FPGA.

from the baseline architecture presented in Section 3.1.2. The NI used as the basis for the experiments is already equipped with rate limiters and translation tables allowing memory mapped NoC address resolution. The NI extensions required by clients follow the design from Section 4.2.3. The evaluation has been concluded by Spieker in [131]. The synthesis results are given in Table 5.3. The table lists all NI modules which have been extended to add a basis for client functionality. The changes include the address translation table (for resolution of a local address into NoC routes) (atranslo), the packetizer (packetizero), the AHB slave interface of the NI (slv_ifo), and the client itself (resbrokero). For the sake of comparison, the table also reports results for higher-level modules, e.g., the whole network interface (netifo) without the client functionality as well as the entire MPSoC system (idamc). It is easily visible that in comparison to other modules of the NI, clients, exhibit a low demand for resources of the FPGA (1% of NI). The detailed evaluation is available in [131].

Modul	Slices	Slice Reg	LUTs	LUTRAM	BRAM FIFO
idamc	43.180	66.559	82.636	76	313
netif0	2.761	3.328	6.033	0	3
atransl0	73	12	132	0	1
packetizer0	254	17	447	0	0
slv_if0	397	621	1.051	0	0
resbroker0	30	70	79	0	0

Table 5.3: Synthesis results of a client as an extension of the network interface on a Virtex-6 LX760 FPGA, from [131].

Moreover, in many setups, the usage of the RM may save resources which would have been assigned to assure real-time properties of synchronized senders. For example, Figure 5.28 presents the area overhead resulting from the implementation of RM and SPP arbitration performed locally in routers in NoCs for different sizes and number of employed priorities. Results are based on the implementation and synthesis of the IDAMC platform with routers offering five flit buffer per VC per ingress port. For small NoCs with only a few HRTTs, the overhead of SPP and

RM are comparable, e.g., 3600 vs 3200 LUTs for four priorities (VCs). However, in case of larger designs, the RM reduces the area overhead significantly, e.g., for a design with eight priorities (VCs) and 32 routers, the area required by the RM is three times smaller than in case of SPP.

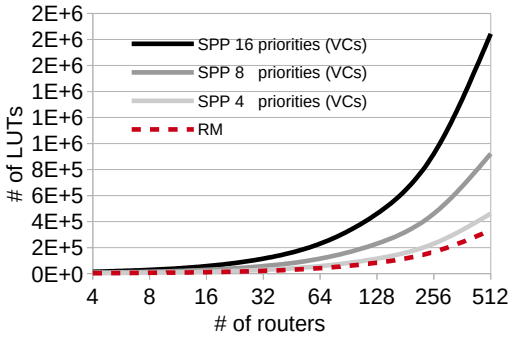


Figure 5.28: Area overhead resulting from the RM in comparison with SPP done locally in routers, based on [75].

HW/SW Co-Design

Finally, the RM unit and client can be implemented in software by re-using the existing platform components. The presented results are based on the specification from Section 4.1.3. The clients logic considers the processing of application requests, synchronization with the RM and programming rate limiters provided by the NI (i.e., it does not include functions of the NI such as packetization or addressing). The initial implementation of the control layer was written in C by considering the Leon3 processor supported by the IDAMC architecture. The compilation was done using a standard GCC compiler (v4.7.3), i.e., the compilation process has not been optimized with respect to any quality of service mechanism. The code size for the implementation of clients is approx. 12 Kb. The code size for the RM unit is approx. 20Kb. Additionally, the size of the tables with settings for synchronization scenarios depends on the complexity of the protocol. In the considered scenario, a LUT entry was made of ScenarioID (8bit max 256 senders, consequently SenderID also 8bit) and priority (4 bits) adding up to 20 bits in total. The settings' entry was made of ScenarioID, SenderID, rate control settings(24 bits), route(32 bits) and VC (4 bits) giving the total of 76 bits per entry. This evaluation shows that the proposed control layer is feasible and can be performed at minimal overhead. The memory

overhead is resulting from settings required for synchronization scales linearly with respect to the number of scenarios and number of senders. The runtime of the RM logic depends directly on the size of the RMs' buffers, which is statically configured at design time. Thus, RM the arbitration is performed at constant runtimes. Please refer to Section 5.5.1 for results regarding the execution times of the synchronized software.

5.5 Case Studies with Commercial and Research MPSoCs

In order to present the effectiveness of our approach, two implementations are discussed using two different MPSoCs as a baseline: the commercially available MPPA [40] and the research oriented IDAMC [140]. Both MPSoCs target real-time applications with high performance requirements. Consequently, the QoS control layer is used to synchronize memory transfers as these transmissions allow us to challenge the performance and the scalability of the system [153].

5.5.1 IDAMC - Research Platform

The Integrated Dependable Architecture for Many Cores (IDAMC) is a many-core processor platform developed at the Institute of Computer and Network Engineering of the TU Braunschweig. Its primary purpose is the evaluation and investigation of how multi- and many-core architectures can be used in timing- and safety-critical applications. The IDAMC [140] is a configurable many-processor platform based on a NoC composed of nodes (N) connected in a 2D mesh topology. Interconnect connects up to 64 nodes (NO-N63) composed of a router (R) and up to four tiles (TI-T4) each. The Tiles are based on the open-source IP library from Gaisler [53]. Each tile contains a bus-based (AMBA bus) sub-system, which may include one or multiple processors (up to 16 Leon3 SPARC cores), on-chip memory, a DDR2 memory controller and other peripherals, e.g., interfaces to Ethernet.

The routers are equipped with up to 8 ports (4 neighbors, 4 local ports) and offer support for a variable number of virtual channels (adjustable). The NoC offers a general purpose architecture (i.e., performance optimized) where flexible source routing and wormhole switching are used. Scheduling in routers is performed using round-robin based arbitration (iSLIP protocol [101]) or static priorities assigned to separate virtual channels. To support Quality-of-Service, nodes are equipped with fine-grained traffic shapers (rate control). For safety-critical communication, IDAMC applies scratchpad memories and DMA engines imposing a large granularity of transfers. Consequently, the IDAMC NoC architecture is effi-

ciently analyzable with the CPA framework using the method from [44],[142], and therefore can be used for implementation of the control layer. The implementation of the control layer in the IDAMC can follow in hardware, software or as hardware/software co-design, due to availability of its HDL sources and FPGA deployment.

Multi-threaded application: ADA

The evaluation of the control layer on the IDAMC platform was done with the Artificial Demonstrator Application (ADA) use-case provided by Airbus within the scope of the EMC2 project. The results has been reported in [26] and the presented evaluation is based on this document. It mimics the behavior of a Helicopter Terrain Awareness and Warning System (HTAWS) application which provides a comprehensive, map-based overview of the helicopters' surroundings in order to prevent avoidable collisions with ground or obstacles. Note, that it's functionality is analogous to an automotive driver assistance system. Its main goal is to support steering of the aircraft in the following conditions: flying at night, changing weather with poor visibility, rough terrain or at low altitudes.

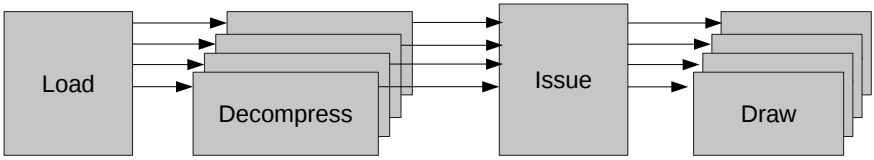


Figure 5.29: Main processing pipeline of the Artificial Demo Application (ADA) from Airbus mimicking the behavior of Helicopter Terrain Awareness and Warning System for helicopters, based on [26].

The ADA is built from 4 to 36 periodically activated threads. These threads construct a single frame with a period of 67ms, designed for execution in a symmetric multiprocessing (SMP) system. The single frame is composed of a variable number of internal minor frames which are not synchronized. The main application (data) flow comprises 4 major pipeline stages (see Figure 5.29). Two stages (*decompress and draw*) can be further parallelized (data parallelism). The depth of these two parallel stages can range from 1 to 16 threads each as the parallel operations are independent. For the reference implementation provided by Airbus, all 4 stages are executed sequentially on the singe core with a total period of 60ms. However, the execution of the stages can be further pipelined for increasing the overall throughput.

However, a careful evaluation of the application shows that only a further parallelization of stage (2) can lead to a speed-up of the application. This is because the processing time of each pipeline stage is highly unbalanced, i.e. stage 2 consumes most of the processing. Due to the imbalance between the processing time in pipeline stages, the operation of the decompression stage has been spread over multiple tiles, when porting the ADA to a multi-tile IDAMC setup. All other remaining stages have been executed on the same single tile. Figure 5.30 depicts this scenario for a 4-tile instance of IDAMC. In the figure, T0 (Tile 0) executes the code for pipeline stages 1, 3 and 4, while T1, T2 and T3 are responsible for processing pipeline stage 2.

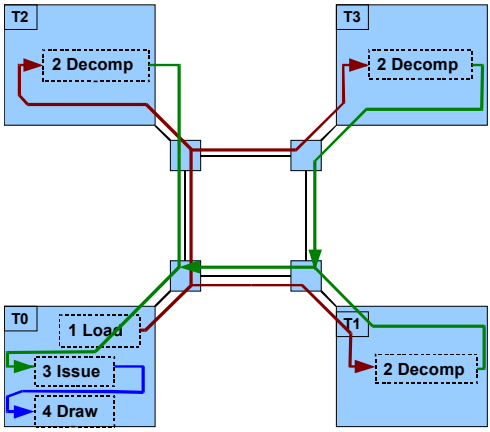


Figure 5.30: Placement of stages on tiles and data flow through the NoC. In the figure, T0, T1, T2 and T3 are IDAMC tiles, based on [26].

Notice that, depending on the size of the input being processed by the real HTAWS application, pipeline stage 2 could be distributed over more than three processing tiles. Moreover, regardless of the number of tiles over which pipeline stage 2 is distributed, this implementation strategy requires chunks of data to be transferred over the network. For instance, in Figure 5.30, the data produced by the first pipeline stage (Load) must be sent over the NoC from T0 to T1, T2 and T3. Similarly, the output of the second pipeline stage (Decompression) must be sent over the NoC from T1, T2 and T3 back to T0.

Finally, in order to synchronize the operation of different pipeline stages, a barrier synchronization function is needed. This will be provided by a simple, shared-

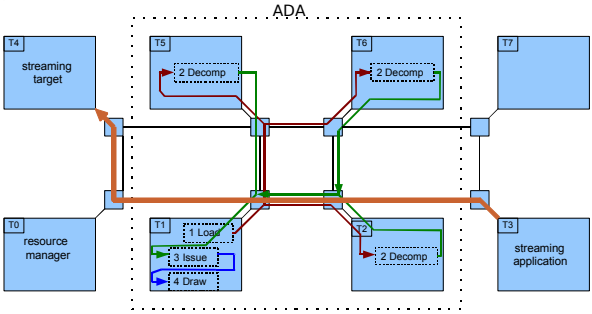


Figure 5.31: IDAMC experimental setup for testing ADA with control layer, based on [26].

memory based implementation built up from two-processor barriers called the "Butterfly Barrier" [28]. This approach works with minimal resources (only a few shared memory words are needed) and without adding any data traffic to the NoC when a processor is actively waiting for the barrier to be reached by the other processors.

Setup and Metrics.

While the data processing of all pipeline stages happens within a tile and is therefore independent of applications running elsewhere in the system, data transfers between the main tile and the three processing tiles for the Decompression stage are dependent on the availability of the NoC. For example, consider the scenario depicted in Figure 5.31, in which an application running on T3 ("streaming app") needs to regularly transfer data to T4 ("streaming target"). Notice that the path that must be traversed for a NoC packet sent from T3 to reach T4 overlaps with the path that tiles running the Airbus Application use to communicate with each other. This means that the performance of ADA can be damaged by the streaming application.

Therefore, to protect the safe and efficient execution of the use-case, the control layer is used with the setup presented in Figure 5.31. In order to evaluate the temporal isolation properties of the proposed control layer, a total of four tiles are used to run ADA. Three tiles generate interference and one tile executes the RM. Two pairs of scenarios are investigated (a pair refers to a set of two executions: one in which requestors use the NoC without supervision, and one in which the requestors negotiate access to the NoC with the RM). As for performance metrics, the following is considered:

No interference	With interference		
10 Batches Latency	Interference Size	10 Batches Latency (without RM)	10 Batches Latency (with RM)
14,545 ms	8KB	8,302 ms	9,338 ms
	128 KB	14,545 ms	9,182 ms

Table 5.4: Temporal isolation evaluation: latency to process 10-batches of data, based on [26].

- The latency observed in order to process 10 batches of data, i.e. the amount of time required for 10 batches of data to go through each of the 4 pipeline stages of ADA. Each batch of data is 200 words.
- The latency of individual data transfers.

For the evaluation, IDAMC is prototyped on a Virtex-6 FPGA. The chosen IDAMC topology is the one depicted in Figure 5.31, in which a total of 8 tiles are present, organized in a 2x4 mesh. The arbitration in routers follows the principles of the iSLIP protocol [101]. The design works with a clock of 80 MHz. Moreover, the NoC from IDAMC is configured to have 2 virtual-channels, one for data transfers and another one for control information (only used if the access control layer is employed). The buffer queues can store up to five flits.

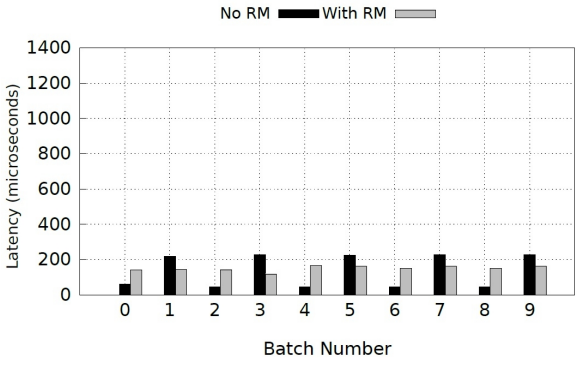
Results

In the first series of experiments, the interfering requestors make transfers that are 8 KB long. In the second, the transfer size is increased to 128 KB. The latency to process a total of 10 batches (each with 200 words) of data is measured and presented in Table 5.4. The table summarizes the results obtained running ADA in four processing tiles without the presence of interference (and without the use of the control layer).

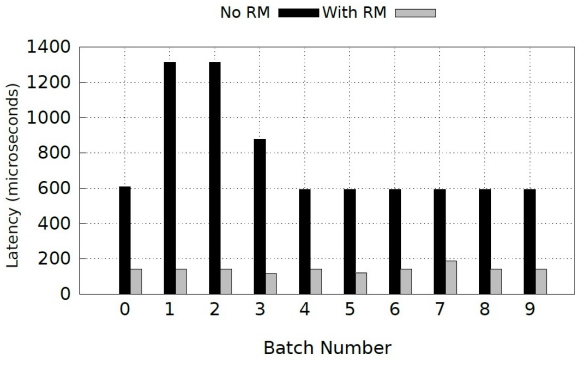
If no interference is present (leftmost column), the latency to process 10 batches of data is of only 5,815 ms, which is significantly smaller than the values observed if interference is present. In the presence of interference and without a NoC access control layer (third column), the latency to process 10 batches of data depends drastically on the size of the transfers performed by the interfering requestors. This is because, as discussed in the previous section, each transfer is sent as a single data-chunk in the NoC, which blocks other packets trying to use the same virtual channel. In the presence of interference, but with a NoC access control layer (rightmost column), the latency to process 10 batches is independent of the size of the transfers from the interfering requestors (although a small difference is seen, it is considered negligible). This is because the NoC access control layer preempts the interfering requestors when ADA wants to use the NoC. Finally, it is worth noticing

that, even though the RM has preemption capabilities, control messages between requestors and the RM must be exchanged, which takes extra time.

Consequently, ADA executes slower in a system with the RM than in a system with no interference (and no RM), i.e. the values in the rightmost column are larger than the values in the leftmost column. Considering the investigated scenarios, it



(a)



(b)

Figure 5.32: Worst case guarantees for a burst of 16 transmissions with jitter =10%P for the ADA benchmark running on the IDAMC architecture (a) Transmission latency (b) Protocol overhead resulting from RM, based on [26].

is also interesting to evaluate the latency of individual data transfers. In order to be concise, this evaluation focuses on the latency of data transfers performed by Tile1 to Tile2 in Figure 5.31 (Tile2 implements the decompression stage of the processing pipeline, and receives its input from Tile1). The measured results are depicted in

Figure 5.32(a) and in Figure 5.32(b). To facilitate a comparison, both figures employ the same scale. Notice that one transfer is required per each batch of data. Hence, for each scenario, the latencies of 10 transfers are shown. In a system that relies on a NoC admission control layer, the latency of the transfers from ADA behave pretty similarly, regardless of how large the interfering transfers are (notice how the gray bars from Figure 5.32(a) are similar to the ones from Figure 5.32(b)). In summary, the use of a RM and the control layer enforces predictable transfer times. In a system without a NoC admission control layer, the transfer times for ADA (black bars) can vary drastically, depending on how much competition is observed in the NoC, i.e. whether an ADA packet is stalled because a packet from the interfering requestors is being transferred. Moreover, as expected, the latencies are significantly higher if the interfering requestors make 128 KB transfers.

Furthermore, the results from the software implementation of the control layer have proven that although the scheme follows the design principles of Software Defined Networks (SDN) these principles are not naively transferable to the embedded real-time domains. The implementation entirely in software results in excessively high synchronization latencies and the dynamic control must be applied to processing nodes rather than routers. For instance, high latencies (milliseconds in results from Table 5.4 and Figure 5.32) are driven by two main factors: custom solutions resulting from the research oriented IDAMC design (e.g. stacked controllers (AMBA, NoC), and low frequency of the processors 100MHz) as well as high processing time of the software stack. The results have confirmed that straightforward implementation of the SDN paradigm for NoCs violates the properties of synchronized workloads (e.g., short latencies of memory accesses) and can be applied only for some selected setups with low synchronization frequency. Recall, that the introduced protocol overhead remains constant with respect to the transmission latency (i.e. duration of synchronization), therefore can be acceptable if synchronizations do not happen frequently. Finally, the control layer provides the solution for formal verification of safety-critical systems which is not a case for SDNs.

5.5.2 KALRAY MPPA - Commercial Platform

The initial evaluation is done considering the MPPA2-256 (Bostan) architecture designed and commercialized by Kalray. The Multi-Purpose-Purpose Processing Array (MPPA) integrates 256 processing engine (PE) cores and 32 resource management cores on a single 28nm CMOS chip. These cores are divided into sixteen computing clusters and four I/O subsystems, see Figure 5.33.

Each of the I/O sub-systems contains a DDR-SDRAM memory (with access to up

to 64GB external DDR), one Ethernet controller, one PCI Express (PCIe) controller, 64 GPIOs, two quad-cores with shared D-cache, an on-chip memory of 4MB, and other I/O devices [40].

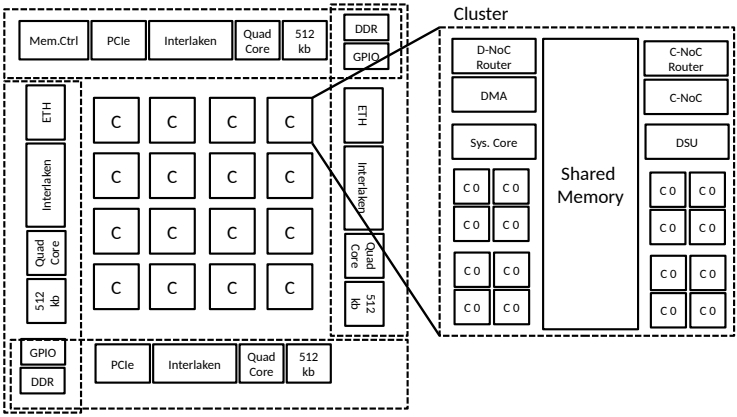


Figure 5.33: The block diagram of the MPPA2-256 (Bostan) architecture, based on [71].

The MPPA provides two distinct physical network layers: a control NoC (C-NoC) and a high-bandwidth data NoC (D-NOC) for DMA transfers. Both NoCs support full duplex links (4B/cycle), wormhole switching, 5-port NoC routers, and output buffering without virtual channels. The FIFO buffer in the router is 401 flits deep for each queue in the D-NoC and eight flits deep for each queue in the C-NoC. The 2D torus topology, see Figure 5.34, connects 32 routers. Sixteen routers (appropriately nodes 1-16) are connected to a processing node (cluster) whereas the remaining ones are connected with I/O peripheral banks on the top (T1-4), bottom (B1-4), left (L), and right (R) edges of the chip.

The MPPA routers implement a round-robin arbitration and the nodes offer rate regulators (rate limiters) with fine-grained traffic control, cf. Sec. 3.6.5. This flow regulation is adjusted for the formal verification done with the network calculus [89] framework. The network calculus establishes a set of linear constraints on the link bandwidths ("capacity constraints") and on the router FIFO queues' sizes ("backlog constraints"). Consequently, the admission control is done for each network node independently and controlled by a packet shaper and a traffic limiter in tandem. These units allow adjusting the bandwidth quota as well as a maximal payload for each regulator per temporal window which is global for the whole D-NoC. The QoS guarantees rely on the absence of buffer overflows in the NoC

routers [41]. Therefore, the guarantees directly depend on the number of the simultaneously running senders and the amount of transmitted data.

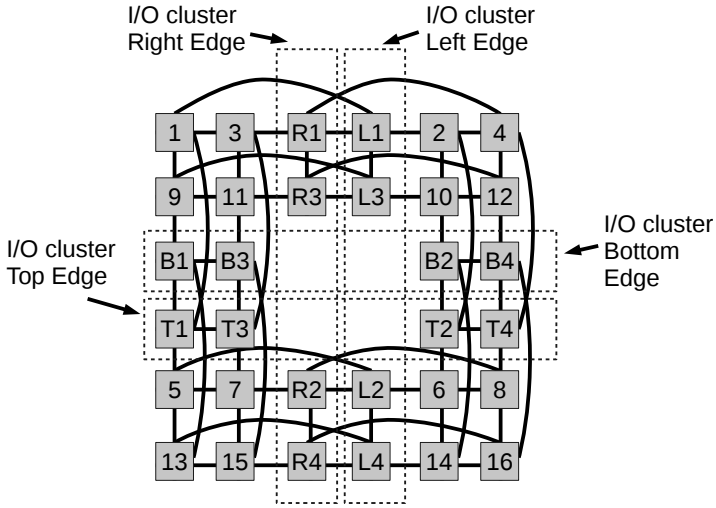


Figure 5.34: 2D torus topology of the C-NOC and D-NOC interconnect layers used by the MPPA architecture, based on [71].

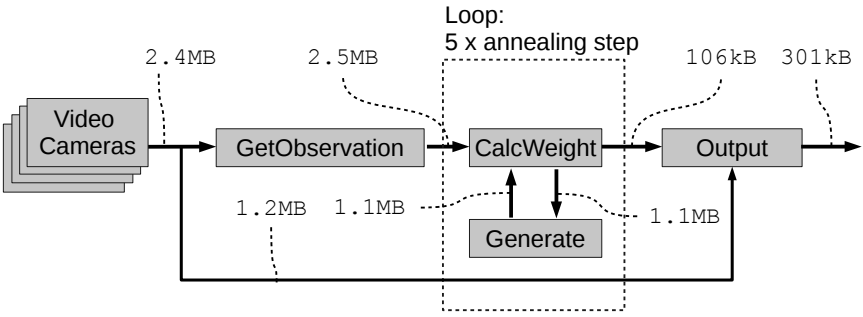


Figure 5.35: Dataflow of the PARSEC bodytrack benchmark, based on [20] and [149].

Bodytrack application from PARSEC benchmark.

The evaluation is done using the *bodytrack* computer vision application from the PARSEC benchmark [20]. The presented analysis of the benchmark is based on the results from [149]. Bodytrack provides a workload from the computer vision application designed by Intel. The benchmark applies an annealed particle filter for tracking 3D-pose or movement of a markerless human body [13]. Its behavior is similar to modules foreseen for autonomous driving - Advanced Driver Assistance System (ADAS) - where a car relies on computer vision for its real-time interactions with an environment without any available aid, e.g., constrained behavior.

Figure 5.35 presents the data flow for the *bodytrack* application. The recorded video frames are stored in the main DDR-SDRAM memory. Consequently, the "GetObservation" module reads four original camera images and four foreground maps. The experiments used the Parsec's native input set of frames: 4 cameras, 261 frames, 4000 particles and five annealing layers. Each frame has a 640x480 pixels resolution and corresponds to 300kB per frame, which makes a total of 2.4MB of input for the "GetObservation" module. Later, the results from the "GetObservation" module are provided to the "CalcWeight" module. They contain four edge maps and four foreground maps (2.5MB). For each of the five annealing steps, Generate module transfers 4000 new particles to the "CalcWeight" module (1.1 MB). Next, results from "CalcWeight" are provided back to the "Generate" module (same 1.1 MB). Once the annealing of the frame finishes, the "CalcWeight" module produces data to the "Output" module with the size of 106kB. Finally, the "Output" module fetches the four original frames (1.2MB) and produces an output image of 301kB with a marked position of the body.

The stages of the *bodytrack* benchmark (GetObservation, CalcWeight, Generate, Output) operate with the Pthreads library. Consequently, a further parallelization with stages is possible, e.g., computing gradient magnitude and threshold, applying Gaussian blur when creating edge map, generating new particle, and calculating particle weight. The multithreading follows the map-reduce principles: the main thread distributes the workload among worker threads and resumes when it receives the results from all independent computations.

Setup and Metrics

This section presents an exemplary deployment of a *BODYTRACK* functionality from the *PARSEC* benchmark in the MPPA using the control layer. The description is based on [149] which contains further details. The limiting factor for the deployment of the benchmark is the amount of the data and instruction memory which is available per cluster. Consequently, the presented solution is done under

the assumption that each cluster has at least 2MB of internal (on-chip) memory. Figure 5.37 presents the mapping of the Bodytrack modules as well as the RM unit.

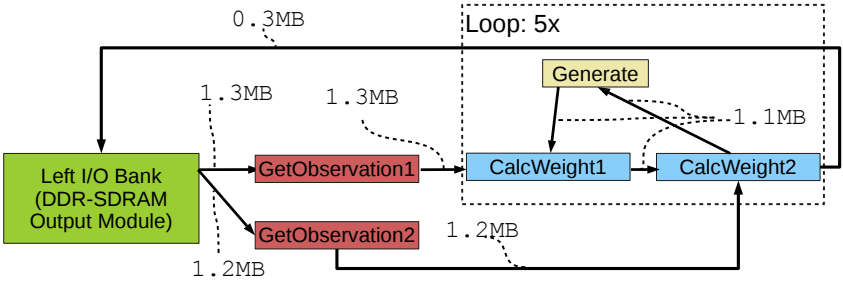


Figure 5.36: Workflow of the PARSEC's bodytrack benchmark adjusted for the 2MB scratchpad memories in clusters, based on [20] and [149].

The operations of modules "GetObservation" and "CalcWeight" are divided into two parts for using available scratchpad memories, as presented Figure 5.36. Consequently, during each computational cycle of **bodytrack**, the "GetObservation1" module requires 4 frames and the remaining class "TrackingModel" (1.3MB). Later, the "CalcWeight1" module uses "TrackingModel" class with four edge maps. The "GetObservation2" module receives four foreground maps (1.2MB). After finishing its operation, the "CalcWeight1" module transmits them to the "CalcWeight2" module. To execute each of the annealing steps, the "CalcWeight1" module receives 4000 new particles from the "Generate" module. Later, the "CalcWeight1" module uses them for computation along with the input from the "GetObservation1".

The "CalcWeight2" module receives the interim results (1.1MB). The "CalcWeight2" module processes the 4000 particles along with four foreground maps from the "GetObservation2" module. Its output is provided to the "Generate" module. After annealing is finished, the "Output" module receives 106kB data from the "CalcWeight2" module. Finally, the "Output" module may generate and write to the DDR SDRAM memory the final image (300kB). This operation requires four original frames from the "GetObservation2" module and the input from the "CalcWeight2" module.

In order to deploy the *bodytrack* benchmark the upper half of the MPPA architecture is used, see Figure 5.37. The main mapping goal was to minimize distances between the modules. Both "GetObservation" modules are placed one hop from the left I/O sub-system. After consideration of the annealing steps which must

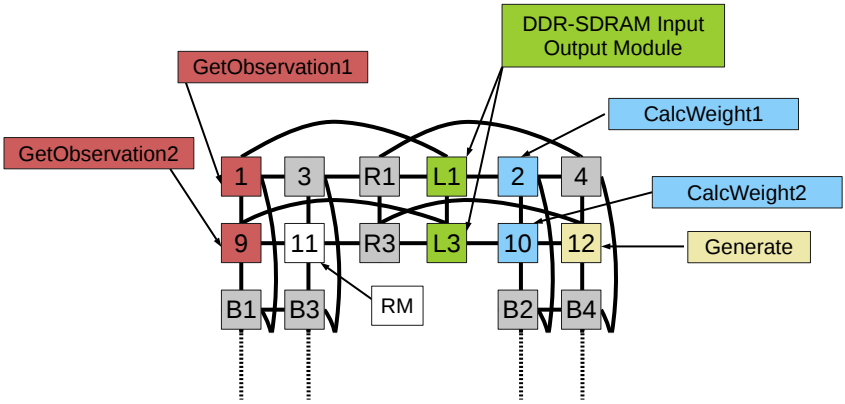


Figure 5.37: Mapping of bodytrack modules to the MPPA nodes, based on [149].

be conducted in a loop, the "CalcWeight1" and "CalcWeight2" modules as well as the "Generate" module are mapped in the nearest proximity. Moreover, the "CalcWeight2" module has a direct connection to the left I/O sub-system to provide the data to the "Output" module. The results from both of "GetObservation" modules are stored intermediately in the DDR SDRAM. Later, the "CalcWeight1" and "CalcWeight2" modules can read this data directly (notice one hop distance). This also helps to keep the executions of the "GetObservation" and "CalcWeight" in order. The Output module is running on a quad-core processor in the left I/O system connected with the node "L2". Due to the full duplex lines on each MPPA link, the transmission from the I/O sub-system from and to modules do not interfere. The proposed mapping in Figure 5.37 depicts only one exemplary solution, as 2D topology offers rotational symmetry.

Clearly, integration of the QoS control plane should, at best, not add to the complexity of the system during operation for both cost and performance reasons. Due to the ASIC deployment of the MPPA, the software implementation of the client and the RM has been considered. The resulting design follows the principles of HW/SW co-design implementation of the control layer which was described in the previous chapter. For example, the considered centralized setup for the QoS control plane consists of clients and the RM. Processor clusters in MPPA architecture are equipped with special supervisor/"high-performance" nodes (used for system's boot-up and configuration) which could be used for hosting and accelerating the main RM's logic as well as clients. The initial stand-alone software library was

programmed in C using the GNU-toolchain, supported by MPPA, with standard libraries (uClibc, Newlib). The RM required between 12kB and 128kB depending on the number of synchronized applications and the complexity of the use-case. Clients required between 12kB and 48kB depending on the use-case. As the complexity of the central RM unit is higher than a client, it can be implemented in the form of a stand-alone processing node, therefore it is mapped to the Cluster 11. Note that the pure benchmarks' functionality has been compiled as a barebone C library with a standard GNU toolchain (gcc ver 4.7.3) . Therefore, its size may change depending on the form of deployment on the MPPA. The control messages were transmitted on the independent, physically separated, control NoC (C-NoC).

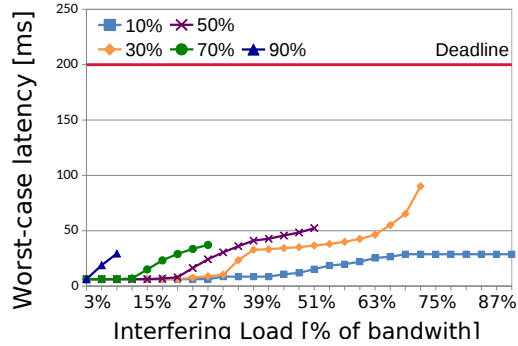
To evaluate the temporal isolation properties of the proposed control layer a formal worst-case analysis of the architecture has been conducted. The developed framework accounts for wormhole switching and considers the effect of pipelining and parallel transmissions. The details of the implementation for MPPA are available in [149]. The following setup of the MPPA architecture has been used for evaluation: flit size: 32 bits; maximum payload in flits per packet: 32; protocol overhead per packet: 2 flits; link bandwidth: 2.4GB/s; constant router delay: 2ns; buffer size: 12 packets. Two pairs of scenarios are investigated: one in which requestors use the NoC without supervision, and one in which the requestors negotiate access to the NoC with the RM. For evaluation purposes it is assumed that bodytrack modules are safety-critical (i.e., have hard real-time requirements) and must not lose any data as it would happen in case of ADAS modules. Consequently, the goal is to evaluate guarantees for bodytrack functions considering different levels of load from interfering senders. These interfering senders are other applications (BE and soft-real time) sharing links with the considered critical modules. The load assignment for BE and SRTs is enforced with rate-regulators. Note, that such resource sharing can happen in MPPA as bodytrack occupies less than 30% of the available processing nodes on chip. As metrics, the worst-case backlog (worst-case number of packets in routers buffer queue) and worst-case latency of a transmission have been selected.

Results Evaluation

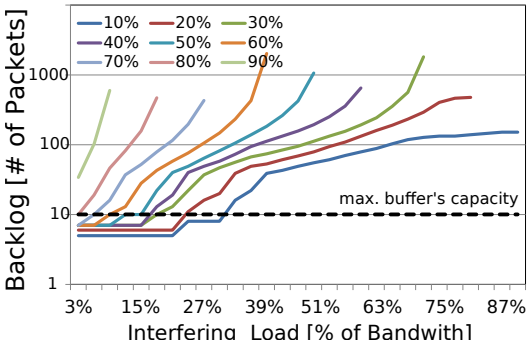
The MPPA is efficiently analyzable with the network-calculus framework [41] or with the SLA framework [149], therefore capable of providing inputs for the admissibility tests. Consequently, for selected, accumulated load values (from all synchronized senders), it is possible to get the upper-bound on the worst case latencies, if they do not exceed the available link bandwidth.

For instance, Figure 5.38(a) presents the worst-case latency for the "GetObservation" module. The results are based on the work of Wang reported in [149]. The

"GetObservation" module's bandwidth requirements have been scaled depending on the frequency and resolution of frames from video cameras constituting the main input (from 10% to 90% of link bandwith). It is visible, that in most sce-



(a)



(b)

Figure 5.38: Worst case guarantees for a burst of 10 transmissions (a) transmission latency (b) backlog in an MPPA using rate limiters. Results are based on [149].

narios the worst-case latency is below 50ms, which is acceptable when compared to the assumed deadline of 200ms. Consequently, the 2.4GB/s link offered by the MPPA is more than enough for the selected setups.

However, these formal Quality-of-Service guarantees rely on the absence of a FIFO queue overflow in the NoC routers as discussed in [41] and [149]. Therefore, they directly depend on the number of simultaneously running applications and the amount of sent data. The static resource allocation enforced by rate regulation

can control only the latter. Therefore, the guarantees are either overly pessimistic or cannot be given i.e. there is an upper bound on the number of applications sharing a path. Figure 5.38(b) presents a worst-case backlog analysis conducted for an exemplary deployment of a BODYTRACK module from the PARSEC benchmark. In the considered scenario, along with the increasing requirements of a sender (resolution and frequency of video frames as % of bandwidth) and the load from interfering senders, the backlog also increases. Consequently, in all scenarios, an interfering load higher than 30% of the bandwidth capacity leads to a violation of safety guarantees.

The next series of experiments investigates the effects of backlog in detail. Figure 5.39(a) presents a comparison of worst-case guarantees for an exemplary deployment of a BODYTRACK module from the PARSEC benchmark sharing a buffer in setups with different NoC load induced by a varying number of interfering senders. For each series of experiments, 1000 different scenarios have been generated with interfering senders placed maximum four hops from the target DDR3 controller. Results depict the percentage of schedulable scenarios, i.e. scenarios in which all the transmissions finish before their deadline.

Consequently, in MPPA along with the increasing accumulated NoC load from interfering transmissions the backlog also increases and the guarantees in most considered scenarios cannot be given, i.e., not schedulable setups. For the further details of evaluation please refer to [149].

In contrast, deployment of the RM results in an significant increase of the utilization by switching on and off selected senders, thus the RM keeps the backlog always on an acceptable level, see Figure 5.39(b). The introduced overhead, in the considered scenario, was always below 5% of a transfer duration (RM placed three hops from sender). Moreover, the RM may decrease or increase injection rates for a particular node dynamically depending on the number of concurrently active applications. The mechanism is capable of enforcing symmetric and non-symmetric guarantees. In the former, transmission rates decrease uniformly for all applications belonging to a synchronization scenario along with the increasing number of concurrent senders. In the latter, transmission rates also depend on the application's importance and may differ between senders. The non-symmetric mode can be used in a mixed-criticality system to maintain guarantees of the critical applications through limiting best-effort traffic, see Section 3.6.5. The presented results are based on [84].

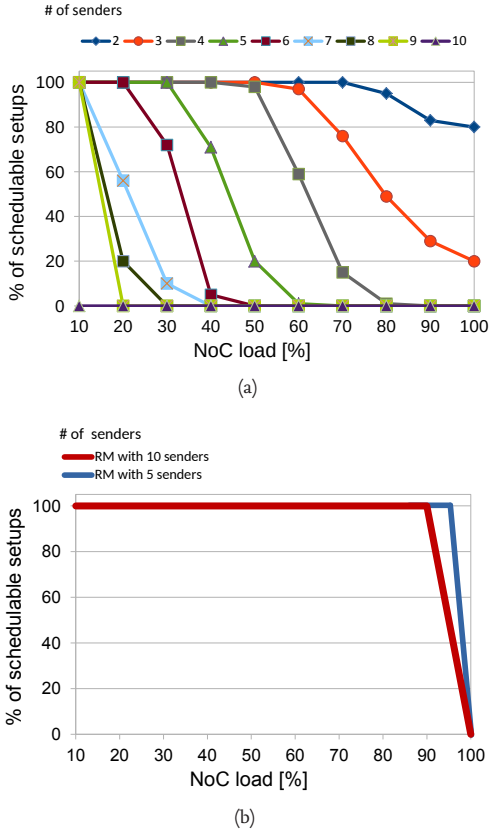


Figure 5.39: Percentage of schedulable scenarios (out of 1000 runs) for an MPPA running (a) without RM (b) with control layer, based on [84].

5.6 Evaluation against requirements

Section 1.4 described the set of functional and non-functional requirements which NoCs should support to host workloads from real-time and/or safety-critical domains. The next paragraphs provide a detailed discussion of how to achieve these goals with help of the control layer. The presented conclusions are supported with the experimental results from the previous chapter. A summary with detailed evaluation of the mechanism with respect to the initial requirements is

presented in Tables 5.5 and 5.6 which include a comparison with other real-time mechanisms for NoCs.

R1 Traffic Types

R1 refers to support for multiple classes of real-time traffic, e.g., GL, GT, BE. This goal can be achieved with the control layer because:

- the control layer can implement different arbitration methods for different synchronized senders, groups of senders, or even selected nodes in the NoC (parts of the network)
- clients are capable of providing fine-granular support for different admission control mechanisms (rate limiters, address translation tables) which allows treating differently the short (cache-based) and long (DMA-based) transmissions
- the mechanism permits flexible, efficient and safe incorporation of scenarios where multiple different traffic types share the interconnect. Different resource allocation schemes can be applied to accommodate these traffic classes without the need for modification of underlying router architectures.

R2 Workload Integration

R2 demands from NoCs to provide efficient support for real-time requirements without the need to modify legacy code and other IPs. The proposed mechanism achieves this goal as the synchronization between the RM and clients can be fully transparent to the legacy code and be implemented as an extension of the underlying NoC infrastructure, e.g., extensions of the NIs. Note, that the accuracy and granularity of synchronization constitutes the main trade-off, as discussed in Chapter 3. Clients must distinguish between different transmissions which requires synchronization with the RM. In case of the coarse grained solution, the client's arbitration is applied to the accumulated workload (traffic from all senders running on this tile). This keeps client complexity at a relatively low level as there is no need to recognize and store information about individual transmissions/connections initiated in the tail. The fine-grained integration of workload requires distinguishing between initiated transmissions from individual senders. This can be achieved through extensions of the address translation units (similar to MMU/MPU) in the NI. However, to achieve the highest efficiency some adjustments of the code running on core may be necessary. An example, would be a new system call which would allow the client to precisely identify a sender (e.g. an OS would write the

appropriate value to the clients register). Note, that such extensions will usually require only minor extensions of OS/drivers for tasks running on the tile.

R3 Flexibility

The proposed solution achieves flexibility of the design through support for different resource allocation strategies depending on a particular setup. As discussed, switching between different strategies for resource allocation in NoCs is relatively simple with the control layer. It requires only reprogramming of the RM module which for this purpose can be delivered in software or as a microcoded processor. It is also possible to apply different resource allocation policies within regions of the same NoC. For instance, for selected nodes static priority based scheduling can be applied whereas for other regions strict temporal isolation of senders with TDM can be applied. The arbitration can be fine-granular and limited to a specific set of resources (e.g. path through the NoC, virtual channel) or mode of system's work (e.g. performance optimized for regular work and priority based for emergency situations).

Finally, the application of different resource arbitration strategies does not require modifications of routers. Therefore, it is possible to offer one chip with a generic router architecture and switch-on the real-time / safety features only in case of concrete deployments, i.e., offer safety and real-time as a feature for a specific design.

R4 Dynamics

The mechanism handles efficiently and safely dynamics in the system's behavior. Firstly, it offers support for work-conserving resource arbitration schemes. Therefore, the workloads are processed as soon as they arrive (or NoC resources are released) minimizing the penalty for senders with variable transmission times or changes in the amount of transmitted data. Furthermore, our solution allows the deployment of a sophisticated contract-based QoS provisioning without introducing complex and hard to maintain management schemes, known from static arbiters. Consequently, the QoS control plane dynamically adapts the NoC to the changing system behavior, mode or environment, which can be influenced by on-chip as well as off-chip factors, e.g.:

- in-field software integrations, including upgrades, software modifications, conditional, or mode dependent functions
- transient errors resulting from technology downscaling raising the susceptibility of the hardware

- self-optimization with respect to aging (temperature), power consumption, response time, or resource utilization
- predictable fail-operational behavior providing for minimum performance overhead error recovery procedures without functional hazard

Recall, that adjustments of scheduling policy can also be done in already deployed chips, as they usually require only a reconfiguration of the RM unit. This permits optimizing, modifying and updating the real-time and safety policy along with updates of running applications (workloads). Such adjustments can be hard or even not possible in case of other QoS mechanisms which are usually integral (hard-coded) part of routers in a NoC. Finally, the control layer may provide self-aware arbitration where the RM adjusts its policy based not only on the predefined scheduling algorithm but also considering the monitoring data (e.g. temperature, access patterns). However, although such approach allows a far-reaching optimization to achieve high performance, it may be hard to certify such systems due to its complexity. Therefore, this challenge is left for future investigation.

R5 Fairness

The control layer allows NoC infrastructure to provide fair allocation of interconnect resources whenever RT senders have different requirements.

Firstly, the proposed mechanism supports different resource allocation schemes, e.g., round-robin, static and dynamic priorities, which could be applied depending on the deployed workloads. The RM has knowledge of the global state of the NoC - which sender is active and which resources are occupied. Using this information, the RM may dynamically decrease or increase the allocated NoC resources (e.g. adjust rate limiters) for a particular node depending on the current system mode. Each mode is set out by the number of currently active applications, and determines the minimum time separating every two transmissions issued from the same application. The mechanism is capable of enforcing symmetric and non-symmetric guarantees. In the former, the amount of allocated NoC resources decreases uniformly for all synchronized senders along with the increasing number of connections (transmissions) running in parallel. In the latter, the amount of allocated NoC resources depends not only on the current system mode but also on the sender's particular requirements (e.g. deadline, slack, priority) and may differ between tiles and senders. The non-symmetric mode can be used in a mixed-criticality system to maintain the critical application guarantees while reducing BE traffic. Consequently, the introduced mechanism enforces behavioral models which are orequired to meet the critical timing constraints of the system for each application at runtime.

R6 Mixed-Criticality

Interfering HRTTs and BEs senders are synchronized using the control layer. This enables the RM to use a dynamic priority arbitration and thus to give BEs access to the NoC whenever there is an available slack and resources are available.

R7 Switch-off QoS

R7 demands from the NoC architecture to provide a mechanism to switch-off real-time and/or safety mechanisms whenever there are no deployed senders with such requirements. Achieving this goal is straightforward with the control layer. The QoS arbitration introduced by the RM and clients is built on-top of the existing routers. Therefore, by switching off clients it is possible to provide the generic NoC functionality. Note, that the hardware overhead resulting from the unused mechanism (clients and the RM) is relatively low and most resources may be released for non real-time setups. Recall that clients are created as extensions of the existing mechanisms in the NIs, thus only small extensions of interfaces and logic are necessary. Moreover, the software deployment of the RM and clients is also possible. In this, case the NI should only provide an interface for controlling its QoS mechanisms (e.g. programming of rate limiters or adress translation tables) for a client running as a task on a processing node (e.g. extension of the OS). Similarly, the processing node used to deploy the RM software can be directly recovered and applied for other purposes whenever there is no need for real-time and/or safety. Note also that the deployment of the control layer may be limited to selected parts (regions) of the SoC.

R8 Scalability

Requirement R8 demands from the NoC architecture to provide performance (average and worst-case) proportionally to the load at runtime (i.e. work conserving arbitration) and not other static factors, such as the number of senders. To do so, the control layer introduces a work-conserving scheduling of network traffic, i.e., it always tries to keep the network resources (links and buffers) busy whenever there are pending connections. This feature along with the capability to run on-top of performance optimized NoC architecture allows overcoming limitations of the strict temporal or spatial arbitration.

However, there are several design trade-offs which one must consider while implementing the control layer directly influencing the scalability of the approach. This is due to the additional latency resulting from the synchronization of the clients and the RM with the selected protocol.

Each control message may interfere with other control messages during trans-

mission or processing. This overhead depends on the frequency, number and latency of necessary synchronizations and re-configurations. Note that, in the considered embedded real-time and/or safety-critical domains, the behavior and characteristics of real-time applications are usually well specified and tested, in contrast to the off-chip networks where the behavior of nodes is often unknown at design time. Therefore, it is possible to derive a load distribution to estimate this interference (i.e. overhead of the synchronization) and assess if response times stay within requested bounds. If the overhead is too high, the designer still has several mechanisms to control it:

- Settings for some applications may be kept on a constant level in some or all modes, i.e., decrease the number of modes of system work. This limits the number of necessary synchronizations and re-configurations.
- In systems where multiple disjoint sets of interfering applications exist, the designer may use different RMs to synchronize each of them independently.
- For maximum performance one may connect the clients and the RM with dedicated lines (star topology). In this case only processing delay must be considered.
- Adjust the granularity of the synchronization. The overhead decreases, as an absolute ratio, with increasing length of connection as the protocol overhead is constant with respect to the transmission/connection duration.
- Decrease the complexity of the protocol or apply another resource allocation scheme.

The latency of control messages is crucial for the performance of the proposed mechanism. In order to minimize the interference with other traffic in the baseline NoC, the control messages are transmitted using VC with the highest priority. More generally, control messages can be allocated to any available VC capable of giving latency guarantees, a dedicated independent control NoC, using an additional bus, e.g. bus-enhanced NoC; or signal lines for maximum performance.

To summarize, the predominant trade-off resulting from global synchronization is between fine granular adjustments (for increasing performance) and both temporal (interference) and hardware (size of clients and RM) overhead.

R9 Locality

R9 demands that the NoC architecture provides the possibility to prevent interleaving of different transmissions. The proposed admission control achieves this

goal through a considered holistic approach: (i) it preserves the locality of memory accesses since the access to the NoC is allocated to the entire transmission (ii) it adjusts the order of granted transmissions (connections) to mitigate the management overhead resulting from the arbiter in the peripheral (interference between the transmissions in the resource scheduler). The RM may introduce a resource oriented arbitration to decrease the performance overhead between the NoC and the memory controller. Furthermore, the RM may improve the (formally proven) worst-case guarantees. A detailed discussion of the interface between the control layer and DDR SDRAM is provided in Section 3.8.

R10 Verification

R10 states that the NoC architecture should support efficient formal verification for standardization/certification purposes (whenever relevant). The control layer achieves these goals, what has been proven in related publications e.g. [81], [78], [83]. In order to provide the predictability of the system, it is possible to compute a bound on the worst case latency for each sender synchronized with the RM. The analysis takes into consideration other interfering senders as well as the overhead of the protocol. The timing relations of the individual transmissions can be abstracted by event models [61] to capture the worst-case and best-case behavior of every possible transmission arrival/activation pattern. Therefore, one may use temporal-analysis frameworks, such as the Compositional Performance Analysis (CPA) [61, 42] (which is applied in this work), to capture the dynamics of the system's behavior with event models.

Additionally, the proposed centralized architecture with the RM permits realizing globally optimal network management based on the current network state. This allows the designer to avoid complex (and potentially lengthy) distributed network control protocols and synchronization scenarios which have to account for the network state before they can take appropriate measures, e.g., propagation of blocking, cyclic dependence. Therefore, the control layer frequently allows providing shorter, formally proven guarantees for end-to-end temporal guarantees in complex SoCs, and or limit the amount of resources (e.g. buffers) necessary for providing such guarantees when compared to other state-of-the-art solutions.

Relevant Requirement	Feature \ Mechanism Type	Performance Optimized NoC (PS, VC, round-robin)	Static Temporal Isolation (TDM)	Dynamic Temporal Isolation (Baseline NoC - PS + VC + RL, static priorities)	RM + Baseline NoC (PS + VC + RL, static priorities)
R1	Quality of Support for GL	low (depends on number of senders and msg. size)	medium (depends on the size of the slots, and synchronization of senders w.r.t TDM-cycle)	medium (depends on senders priority; num. of VCs, and number of priorities)	high (independent of available NoC resources through different allocation schemes may be directly adjusted to requirements of senders)
R1	Quality of Support for GL	low (static isolation depends directly on the available resources)	high	medium	high (possibility to efficiently schedule whole logical transmissions)
R6	Performance of BE senders in mixed-critical scenarios	high (low avg latencies)	low (high avg latencies, depends on the slot size and number of senders)	low (usually lowest priority high. avg. latencies)	high (possibility to apply slack-based arbitration, low average latences)
R2, R3	Sender Penalty for Var. Trans Size	low	high (depends on the slot size and num. of senders)	low	low (support for work-conserving schedulers)
R2, R3	Sender Penalty for Var. Jitter	low (work-conserving arbitration)	high (depends on the slot size and num.of senders)	low (work-conserving arbitration)	low (support for multiple work-conserving schedulers)
R5, R3	Arbitration Fairness for GL Senders	low (depends on the distances and traffic from other senders)	medium (depends on the slot allocation policy)	low	high (multiple schedulers including dynamic priorities and round-robin)
R5,R3	Arbitration Fairness for GT Senders	low (depends on the distances and traffic from other senders)	medium (depends on the slot allocation policy)	medium (depends on available resources)	high (multiple schedulers including dynamic priorities and round-robin)
R4, R3	Performance Penalty for others senders in setups with jitter and var. trans	no	yes (high)	no	no (RM supports several work-conserving schedulers)
R7	Possibility to switch-off QoS	yes	no	yes (but BE workloads may suffer from the high performance penalty)	yes

Table 5.5: Evaluation of the control layer, part 1.

Relevant Requirement	Feature \ Mechanism Type	Performance Optimized NoC (PS, VC, round-robin)	Static Temporal Isolation (TDM)	Dynamic Temporal Isolation (Baseline NoC - PS + VC + RL, static priorities)	RM + Baseline NoC (PS + VC + RL, static priorities)
R7	Hardware overhead in setups with disabled QoS	low (workloads may suffer from high jitter)	high (requires BE NoC or traffic suffers from high avg latencies)	medium (but workloads may suffer from the high jitter and out-of-order arrival of packets)	low (most resources from the control layer can be re-assigned for other purposes, possibility to introduce a round-robin based scheduling on top of routers with SPP)
R8	Support for Scalability	good	poor (depends on the num. of senders and not their activities)	poor (but statically limited to the available HW resources)	good (largely independent of the underlying HW resources e.g. priority-based sharing of the same VC)
R8	Scalability Temporal Penalty	low (but depends on the available HW resources)	high (depends on the num. of senders and not their activities)	low (if there are enough HW resources)	medium (depends on the protocol overhead number and frequency of synchronized reconfigurations)
R8	Scalability Resources Penalty	low (but depends on the available HW resources)	high (requires optimization of the slot allocation e.g. PhaseNoC)	high (a VC per priority level)	low (proportional only to the load and independent of resources)
R9	Support for Locality	no (each individual packet is treated equally thus no protection for longer transmissions)	yes (but requires long TDM slots, which decreases performance in setups with dynamics)	no (yes only for highest prio)	yes (The RM decides the order of the transmissions therefore non-preemptive priority based schedules are also possible)
R10	Pessimism of Formal Verification Static Setups	high (large worst-case scenarios with high interference)	low	medium (depends on the HW resources in NoC, num. of VCs and size of buffers, low if there are enough resources otherwise high)	low (application of the global scheduling allows removing cyclic dependencies, efficiently capture dynamics)
R10	Complexity Formal Verification Static Setups	low	low	low (complexity depends on the amount of available NoC resources low if there are enough resources high otherwise)	medium (complexity depends directly on the complexity of the introduced synchronization protocol)

Table 5.6: Evaluation of the control layer, part 2.

5.7 Summary

This chapter provided the experimental evaluation of the proposed mechanism. The conducted experiments concentrated on the worst-case guarantees and the average performance achieved by senders synchronized with the proposed control layer. As discussed in Chapter 1, these criteria are of critical importance for the future generations of NoCs in real-time and/or safety-critical domains. For evaluation purposes, diverse benchmarks have been used. The experiments include synthetic NoC workloads (for covering corner cases) as well as profiles of real applications (for testing an average behavior). The latter group of the traffic profiles encompass memory traces from the CHSTONE benchmark[51] as well as multiple use-cases: MPEG-4 [19], the real-time video denoising application [95], the automated flight manager from Airbus, and finally, the object detection in the *bodytrack* application from the Parsec benchmark suite. One of the most significant features of the control layer is its flexibility, allowing the straightforward integration in different NoCs. Therefore, tests considered not only the generic NoC with virtual channels [37] (commonly applied in the related research) but also case studies with the commercial (MPPA [40]) and research (IDAMC [140]) oriented MPSoCs.

Experimental results have confirmed the design assumptions from Chapter 3. In the most of the considered scenarios, the control layer was capable of providing a high average performance along with tight real-time guarantees. Therefore, results have proven that the proposed mechanism is capable of reducing performance penalties resulting from other QoS schemes in real-time NoCs. Firstly, the comparison with the TDM-based NoCs resulted in up to 80% of improvement in the considered scenarios using DMA transfers. Similarly, adjusting dynamically the rate control in Nis for handling cache-based traffic resulted in up to 75% improvement when compared to the scenarios with the static arbitration. The mechanism allows supporting multiple scheduling schemes (e.g. time-driven schedulers, static and dynamic priority based arbitration) in the same baseline architecture. There is no need for arbitration specific modifications. Moreover, the RM-based arbitration allows optimization of the NoC schedule for improving the synchronization of the processing nodes with the peripherals connected to the MPSoC. For instance, in case of the DDR3-SDRAM the implementation of the control layer significantly decreased the latency of memory accesses (up to 70% improvement) while providing the real-time guarantees.

The conducted evaluation has also shown that the proposed synchronization can be efficiently applied for transmissions with different granularity levels (e.g. flit, packet, DMA transfers or activations of whole tasks). This introduces a fine-granular control of the overhead (resulting from the synchronization protocol) and

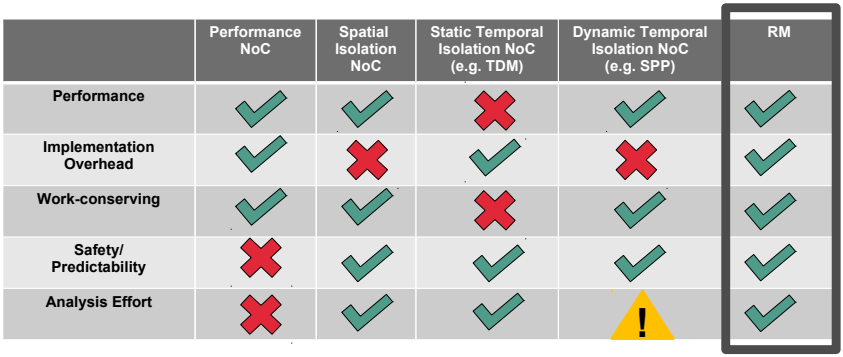


Figure 5.40: Comparison of different QoS mechanisms for NoCs with the control layer. Special focus is placed on safety and performance at the presence of dynamics.

permits keeping it at an acceptable level (e.g. below 5% of the duration of considered NoC access).

Next, the resource overhead resulting from clients and the RM modules has been evaluated. The synthesis reports were conducted based on the implementation in the IDAMC platform [140], using a Virtex-6-LX760 Xilinx FPGA and Xilinx ISE 14.6 with s default optimization settings and no special optimizations for the VHDL implementation. The synthesis results show, that the simple approaches (e.g. round-robin, adaptive load distribution) introduce less than 5% overhead to the area of the NI module. The more complex approaches (e.g. adaptive load distribution) introduce slightly more overhead (5.5%). This slight increase comes from the higher complexity of the control logic which is necessary for an online adoption. However, the control layer simultaneously decreases the hardware requirements of the MPSoC due to the global synchronization and therefore reduce router costs e.g. shorter buffer queues, simpler arbiters. The RM safely limits the size of buffers and head of line blocking without the need for back pressure. Moreover, in the highly dynamic priority-based setups, the control layer can be used to decouple a number of priorities from the available HW-resources, i.e. priority based sharing of the same buffer queue. Sections 5.4 and 5.5 proved that in many systems, the resource overhead from the control layer can be minimized through re-using the existing components of the MPSoC architecture (i.e. different NoC layers for control messages, existing resource management cores). Consequently, the hardware overhead is low and feasible for deployment in many contemporary and future setups. The introduced analysis framework allows not only to provide temporal

guarantees but also, even more importantly, to reach improved service guarantees in many setups when compared to the static resource allocation schemes. The detailed results from the experimental verification are available in the Section 5.2. The improved service guarantees are possible due to the global point of synchronization simplifying the formal verification and capturing the system wide dynamics. Note, that guarantees can be given for the architecture without hard trade-offs between predictability and performance or required HW resources (e.g. buffer size) which are known from other solutions.

Consequently, the control layer is fulfilling all the evaluation criteria, see Table 5.40. The proposed mechanism constitutes a feasible and appealing alternative for the future designs requiring simultaneously high performance and real-time guarantees.

Chapter 6: Conclusions

NoCs designed to host advanced embedded applications require, simultaneously, performance and efficient real-time guarantees in the presence of dynamics. Furthermore, new complex functions, e.g., autonomous driving, combine high load requirements with a need for safety guarantees and therefore must provide a verification methodology to comply with the safety standards.

Existing architectures apply Quality-of-Service mechanisms locally in routers and NIs to address these problems. Their primary goal is to minimize the non-functional dependencies between senders and to put a predictable upper bound on the interference in the interconnect. However, as shown in Chapter 2, this is frequently done at the cost of significantly reduced performance or high hardware overhead. The static platform management, as used in current safety-critical systems, is not sufficient anymore to provide the needed level of service. Dynamic platform management could meet the challenge, but it usually suffers from a lack of predictability and the simplicity necessary for certification of safety and real-time properties.

Solving this major challenge was the main goal of the presented work. In Chapter 3 the novel, *global and dynamic control for NoCs* with real-time requirements has been proposed. The arbitration between concurrent senders is done through the protocol based synchronization between a central scheduling unit - *Resource Manager* - and local arbiters in nodes - *clients*. The behavioral models for senders are enforced with traffic regulators (shapers) in the NIs connecting the node with the NoC. Consequently, the admission control in the NoC is decoupled from the flow control decisions. This allows a path-oriented approach in which both the per-hop behavior of routers and the end-to-end properties of the communication can be unified.

To translate these assumptions into real designs, Chapter 4 provides simulation tools and a verification methodology for setups with different QoS requirements and traffic classes. From a conceptual point of view, the introduced NoC control employs a formal, model-based verification to prove the adherence to constraints

whenever a safe adaptation is necessary.

The results of the evaluation have been presented in Chapter 5. Firstly, the conducted experiments have confirmed that the underlying principles of the proposed scheme allow flexible implementations. Therefore, the solution can be applied to different NoC architectures and MPSoCs, if they comply with some general requirements defined in Chapter 3, e.g., predictable arbitration in routers, adjustable admission control in nodes, a predictable latency of control messages. In most of setups (e.g., research-oriented IDAMC, commercial MPPA platform), only small extensions of the underlying infrastructure were necessary for the incorporation of the control layer, e.g., additional registers in NIs, small client logic. The flexibility of the solution also allows a fine-granular resource control, e.g., implantation entirely in hardware or HW-SW co-design to minimize the overhead.

Simultaneously, the introduced resource management scheme decreases hardware overhead whenever the hard real-time guarantees are necessary. This is possible due to the proposed arbitration based on the global state of the system. For instance, a RM allows to safely limit the load to avoid buffer overflows which could endanger safety (i.e., permits smaller buffers) or to decouple the number of priorities in the NoC from available VCs. Furthermore, the global and centralized arbitration allows handling dynamics in systems' behavior efficiently. As opposed to the frequently applied static arbiters, the sophisticated contract-based QoS provisioning can be implemented with the RM without introducing complicated and hard to maintain schemes or costly hardware extensions. The control layer supports the full range of the available schedulers, e.g., TDM, round-robin, as well as static and dynamic priorities. This allows to flexibly and dynamically adjust the arbitration depending on the system work conditions.

Consequently, the guarantees for the senders with hard real-time requirements can be achieved while simultaneously offering high average performance to BE applications. Indeed, an experimental comparison with TDM resulted in up to 80% improvement in considered scenarios with DMA transfers [81]. Similarly, the RM which is adjusting rate limiters in nodes (for handling cache-based traffic) resulted in up to 75% improvement [78]. The mechanisms offer high flexibility, supporting multiple scheduling schemes (time-driven schedulers, static and dynamic priority based arbitration). The majority of experiments reported improvements in a range from 30% to 70% [81],[83], [75], [78], [77]. The introduced interface to peripheral schedulers' control layer allows decreasing the latency of the memory accesses while providing real-time guarantees e.g., up to 70% improvement in case of DDR3 SDRAM [82].

Although the scheme follows the design principles of Software Defined Networks (SDN), it adjusts them for the requirements of real-time embedded systems,

e.g., automotive and avionics domains. The results from Chapter 5 have confirmed that a naive implementation of the SDN paradigm for NoCs is not possible due to the properties of synchronized workloads (e.g., single memory accesses) and specifics of embedded architectures. The implementation entirely in software results in excessively high synchronization latencies and the dynamic control must be applied to processing nodes rather than routers. Finally, the control layer provides a solution for formal verification in case of safety-critical systems, which is not the case for SDNs.

Summarizing, this work has proposed a mechanism which is capable of achieving temporal predictability without compromising other benefits resulting from the application of NoCs in a SoC, e.g., scalability, modularity, flexibility and excellent performance. Moreover, principles of the control layer are universal and flexible, thus offering a broad scale of possible industrial deployments. It is certain that such synchronization mechanism is an essential step towards an efficient application of NoCs in critical applications with high performance requirements and dynamics such as in automated driving.

However, there are still multiple open questions which provide a direction for future research. Firstly, further synchronization of on- and off-chip traffic could be investigated. This includes communication between multiple RMs in the same system as well as shared resource reservations to provide end-to-end latency guarantees. Moreover, control layer could simplify integration of heterogeneous networks, e.g., running with different frequencies and providing a different protocol granularity. An integration of the SDN-controlled Ethernet within on-chip MPSoC traffic, as foreseen for future automotive setups, seems to be especially interesting. Additionally, the suggested next step is to research on global power management strategies as well as resilience and error control. The changes mentioned above could lead to significant improvements in guaranteed performance and safety for future MPSoC generations allowing a higher density and complexity of integrated functions. Finally, future research could also investigate the possible adjustments of NoC architectures in order to maximize the benefit from the control layer while minimizing the overhead from the synchronization protocol.

List of Publications

This appendix provides the list of all publications written by the author. The list is divided into thesis related and thesis unrelated publications. *All presented works has been peer reviewed.*

Related to the Thesis

1) Adam Kostrzewa, Sebastian Tobuschat, Philip Axer and Rolf Ernst, "Supervised Sharing of Virtual Channels in Networks-on-Chip" in In Proc. of SIES, (Pisa, Italy), Juni 2014.

The article presents the challenges and sketches solutions for global, dynamic and work-conserving arbitration in Networks-on-Chip.

2) Adam Kostrzewa, Selma Saidi, Leonardo Ecco and Rolf Ernst, "Flexible TDM-based resource management in on-chip networks" in Proceedings of the 23rd International Conference on Real Time and Networks Systems (RTNS), vol. 23, (Lilly, France), pp. 151-160 , 2015.

The article presents the control layer protocol for introducing TDM-based resource management in Network-on-Chip. The discussion is supported by formal worst-case analysis.

3) Adam Kostrzewa, Selma Saidi and Rolf Ernst, "Dynamic Control for Mixed-Critical Networks-on-Chip" in IEEE Real-Time Systems Symposium (RTSS), (San Antonio, TX, USA), December 2015.

The paper presents the static priority based resource management in real-time Network-on-Chip using the control layer. The work combines the global, work-conserving scheduling for the end to end guarantees with the local arbitration in routers.

4) Adam Kostrzewa, Selma Saidi, Leonardo Ecco and Rolf Ernst, "Dynamic admission control for real-time networks-on-chips" in Design Automation Conference (ASP-DAC) 21st Asia and South Pacific, (Macau, China), Januar 2016.

The work targets the limitations of the TDM-based network management in NoC. Consequently, it introduces an alternative roand-robin based scheme through a dedicated control layer protocol.

5) Adam Kostrzewa, Selma Saidi, Leonardo Ecco and Rolf Ernst, "Ensuring safety and efficiency in networks-on-chip" , Elsevier Integration, the VLSI Journal, 2016.

The article extend the work from ASP-DAC 2016, by considering memory effect of modern DDR-SDRAM chips. Of special interested are temporal properties and response time in combination with locality of the accesses. Consequently, control layer extensions are proposed for forming an interface between the memory and NoC.

6) Adam Kostrzewa, Selma Saidi and Rolf Ernst, "Slack-based resource arbitration for real-time Networks-on-Chip" in Design, Automation & Test in Europe Conference & Exhibition (DATE), (Dresden, Germany), April 2016.

The work presents the dynamic allocation scheme for BE sender i.e. providing latency guarantees for hard real-time transmissions with minimum impact on performance sensitive best-effort traffic. This is performed using the control layer.

7) Adam Kostrzewa, Rolf Ernst and Selma Saidi, "Multi-path scheduling for multimedia traffic in safety critical on-chip network" in 2016 14th ACM/IEEE Symposium on Embedded Systems For Real-time Multimedia (ESTIMedia), vol. 14, (Pittsburgh, PA, USA), pp. 1-10, Oktober 2016.

The work considers multi-path traversals of safety-critical traffic with real-time requirements in a system supervised with the control layer.

8) Adam Kostrzewa, Sebastian Tobuschat, Selma Saidi and Rolf Ernst, "Supporting Suspension-based Locking Mechanisms for Real-Time Networks-on-chip" in 24th International Conference on Real-Time Networks and Systems (RTNS), vol. 24, (Brest, France), pp. 215-224 , Oktober 2016.

Enabling suspension-based locking requires providing feedback about the global state of the interconnect. For this purpose, in the article the control layer extension is proposed forming an interface between cores and interconnect.

9) Adam Kostrzewa, Sebastian Tobuschat, Leonardo Ecco and Rolf Ernst, "Adaptive load distribution in mixed-critical Networks-on-Chip" in 22nd Asia and South

Pacific Design Automation Conference (ASP-DAC), 2017 .

The publication discusses a protocol-based adaptive load distribution which by selectively detouring BE traffic i.e. load balancing, allows to significantly improve NoC's performance without costly hardware extensions.

10) Adam Kostrzewa, Sebastian Tobuschat and Rolf Ernst, "Self-Aware Network-On-Chip Control in Real-Time Systems", IEEE Design & Test, 2018.

The article discusses the application of the control layer in the context of the new highly complex embedded applications, e.g. autonomous driving, which require simultaneously high performance and safety in highly dynamic setups.

11) Sebastian Tobuschat, Adam Kostrzewa and Rolf Ernst, "Selective congestion control for mixed-critical networks-on-chip", Elsevier Integration, the VLSI Journal, 2017.

The article discusses selective congestion control for mixed-critical networks which is based on the control layer. By selectively detouring real-time or BE traffic (i.e load balancing) and dynamic throttling of BE, the proposed solutions allow improving the NoC performance without costly hardware extensions.

Unrelated to the Thesis

1) Leonardo Ecco, Selma Saidi, Adam Kostrzewa and and Rolf Ernst, "Real-Time DRAM Throughput Guarantees for Latency Sensitive Mixed QoS MPSoCs" in Proceedings of the IEEE 10th International Symposium on Industrial Embedded Systems (SIES), (Siegen, Germany), Juni 2015, - BEST PAPER AWARD.

The work proposes a memory controller allows providing low latency for best-effort requestors and real-time guarantees for senders requiring throughput guarantees.

2) Leonardo Ecco, Adam Kostrzewa and Rolf Ernst, "Minimizing DRAM Rank Switching Overhead for Improved Timing Bounds and Performance" in Proceedings of the IEEE Euromicro Conference on Real-Time Systems (ECRTS), (Toulouse, France), Juli 2016.

This paper proposes a mixed critical real-time controller for multi-rank DRAM modules that minimizes rank switches. The introduced controller works by scheduling batches of data transfers for each rank and performing rank switches only in the end of each batch.

3) Sebastian Tobuschat, Adam Kostrzewa, Falco K. Bapp and Christoph Dropmann, "Online monitoring for safety-critical multicore systems", it - Information Technology, 2017.

The work originates from the ARAMIS research project. It discusses usage and limitations of monitoring approaches in the context of safety critical real-time deployments.

4) Werner Weber, Alfred Hoess, Adam Kostrzewa, Rolf Ernst and others, "The EMC2 Project on Embedded Microcontrollers: Technical Progress after Two Years" in Digital System Design (DSD), Euromicro Conference on, 2016.

The publications discusses the control layer development and other research activities in the context of the EMC2 Artemis project.

Bibliography

[1] Intel many integrated core architecture - advanced. (2011). http://www.intel.com. Accessed: 10.03.2018.

[2] Tile processor architecture overview for the tile-gx series. http://www.mellanox.com/repository/solutions/tile-scm/docs/UG130-ArchOverview-TILE-Gx.pdf. Accessed: 10.03.2018.

[3] Do-254 - design assurance guidance for airborne electronic hardware, April 2000.

[4] Do-178B - software considerations in airborne systems and equipment certification, December 2009.

[5] IEC SC 65A. Functional safety of electrical/electronic/programmable electronic safety-related systems. Technical Report IEC 61508, The International Electrotechnical Commission, 3, rue de Varembé, Case postale 131, CH-1211 Genève 20, Switzerland, 1998.

[6] L. Abdallah, J. Ermont, J. l. Scharbarg, and C. Fraboul. Towards a mixed noc/afdx architecture for avionics applications. In *2017 IEEE 13th International Workshop on Factory Communication Systems (WFCS)*, pages 1–10, May 2017.

[7] L. Abdallah, M. Jan, J. Ermont, and C. Fraboul. I/o contention aware mapping of multi-criticalities real-time applications over many-core architectures. In *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1–1, April 2016.

[8] Niket Agarwal, Tushar Krishna, Li-Shiuan Peh, and Niraj K Jha. Garnet: A detailed on-chip network model inside a full-system simulator. In *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, pages 33–42. IEEE, 2009.

[9] AILab. *http://www-ailab.elcom.nitech.ac.jp/*, Accessed online 29.11.2017.

[10] Benny Akesson, Kees Goossens, and Markus Ringhofer. Predator: A predictable sdram memory controller. In *Proceedings of the 5th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis*, CODES+ISSS '07, pages 251–256, New York, NY, USA, 2007. ACM.

[11] Arteris. *http://www.arteris.com/flexnoc*, Accessed online 1.12.2017.

[12] S. Azodolmolky, R. Nejabati, M. Pazouki, P. Wieder, R. Yahyapour, and D. Simeonidou. An analytical model for software defined networking: A network calculus-based approach. In *2013 IEEE Global Communications Conference (GLOBECOM)*, pages 1397–1402, Dec 2013.

[13] A. O. Balan, L. Sigal, and M. J. Black. A quantitative evaluation of video-based 3d person tracking. In *Proceedings of the 14th International Conference on Computer Communications and Networks*, ICCCN '05, pages 349–356, Washington, DC, USA, 2005. IEEE Computer Society.

[14] Mario Baldi, Silvano Gai, and Gian Pietro Picco. Exploiting code mobility in decentralized and flexible network management. In *Proceedings of the First International Workshop on Mobile Agents*, MA '97, pages 13–26, London, UK, UK, 1997. Springer-Verlag.

[15] Y. Ben-Itzhak, E. Zahavi, I. Cidon, and A. Kolodny. Hnocs: Modular open-source simulator for heterogeneous nocs. In *2012 International Conference on Embedded Computer Systems (SAMOS)*, pages 51–57, July 2012.

[16] L. Benini and G. De Micheli. Networks on chips: a new soc paradigm. *Computer*, 35(1):70–78, Jan 2002.

[17] Luca Benini and Giovanni De Micheli. *Networks on Chips: Technology and Tools*. Systems on Silicon. Morgan Kaufmann, San Francisco, 2006.

[18] Konstantin Berestizshevsky, Guy Even, Yaniv Fais, and Jonatan Ostrometzky. Sdnoc: Software defined network on a chip. *Microprocessors and Microsystems*, 50(Supplement C):138 – 153, 2017.

[19] D. Bertozzi, A. Jalabert, Srinivasan Murali, R. Tamhankar, S. Stergiou, L. Benini, and G. De Micheli. Noc synthesis flow for customized domain specific multiprocessor systems-on-chip. *IEEE Transactions on Parallel and Distributed Systems*, 16(2):113–129, Feb 2005.

[20] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The parsec benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, pages 72–81, New York, NY, USA, 2008. ACM.

[21] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011.

[22] T. Bjerregaard and J. Sparso. Implementation of guaranteed services in the mango clockless network-on-chip. *IEEE Proceedings - Computers and Digital Techniques*, 153(4):217–229, July 2006.

[23] Paul Bogdan and Radu Marculescu. Workload characterization and its impact on multicore platform design. In *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES/ISSS '10, pages 231–240, New York, NY, USA, 2010. ACM.

[24] Evgeny Bolotin, Israel Cidon, Ran Ginosar, and Avinoam Kolodny. Qnoc: Qos architecture and design process for network on chip. *J. Syst. Archit.*, 50(2-3):105–128, February 2004.

[25] Shekhar Borkar. Thousand core chips: A technology perspective. In *Proceedings of the 44th Annual Design Automation Conference*, DAC '07, pages 746–749, New York, NY, USA, 2007. ACM.

[26] Thomas Boroske, Ecco Leonardo, Jan Westermann, Adam Kostrzewa, et al. *Deliverable D.14 Hybrid Avionic Integrated Architecture Detailed Design and Prototype.* WP8 Hybrid Avionic Integrated Architecture. EMC2 Project Consortium, 2017.

[27] Björn B. Brandenburg and James H. Anderson. The omlp family of optimal multiprocessor real-time locking protocols. *Design Automation for Embedded Systems*, 17(2):277–342, 2013.

[28] Eugene D. Brooks. The butterfly barrier. *International Journal of Parallel Programming*, 15(4):295–307, Aug 1986.

[29] A. Burns, J. Harbin, and L. S. Indrusiak. A wormhole noc protocol for mixed criticality systems. In *2014 IEEE Real-Time Systems Symposium*, pages 184–195, Dec 2014.

[30] Alan Burns and Robert I. Davis. Mixed criticality systems - a review, 9th edition. In *Technical Report, University of York, York, UK.*, 2017.

[31] Alan Burns, Leandro Soares Indrusiak, and Zheng Shi. Schedulability analysis for real time on-chip communication with wormhole switching. *Int. J. Embed. Real-Time Commun. Syst.*, 1(2):1–22, April 2010.

[32] Vincenzo Catania, Andrea Mineo, Salvatore Monteleone, Maurizio Palesi, and Davide Patti. Cycle-accurate network on chip simulation with noxim. *ACM Trans. Model. Comput. Simul.*, 27(1):4:1–4:25, August 2016.

[33] S. Chakraborty, S. Kunzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *2003 Design, Automation and Test in Europe Conference and Exhibition*, pages 190–195, 2003.

[34] K. Chandrasekar, B. Akesson, and K. Goossens. Improved power modeling of ddr sdrams. In *Digital System Design (DSD), 2011 14th Euromicro Conference on*, pages 99 –108, 31 2011-sept. 2 2011.

[35] Liu Cong, Wang Wen, and Wang Zhiying. A configurable, programmable and software-defined network on chip. In *2014 IEEE Workshop on Advanced Research and Technology in Industry Applications (WARTIA)*, pages 813–816, Sept 2014.

[36] AUTOSAR Consortium. *AUTOSAR OS, https://www.autosar.org/standards/classic-platform/*, Accessed online 29.11.2017.

[37] William Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[38] William J. Dally and Brian Towles. Route packets, not wires: On-chip interconnection networks. In *Proceedings of the 38th Annual Design Automation Conference*, DAC '01, pages 684–689, New York, NY, USA, 2001. ACM.

[39] R. I. Davis, K. W. Tindell, and A. Burns. Scheduling slack time in fixed priority pre-emptive systems. In *1993 Proceedings Real-Time Systems Symposium*, pages 222–231, Dec 1993.

[40] B. D. de Dinechin, D. van Amstel, M. Poulhiès, and G. Lager. Time-critical computing on a single-chip massively parallel processor. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1–6, March 2014.

[41] Benoît Dupont de Dinechin, Yves Durand, Duco van Amstel, and Alexandre Ghiti. Guaranteed services of the noc of a manycore processor. In *Proceedings of the 2014 International Workshop on Network on Chip Architectures*, NoCArc '14, pages 11–16, New York, NY, USA, 2014. ACM.

[42] Jonas Diemer, Philip Axer, and Rolf Ernst. Compositional performance analysis in python with pycpa. In *3rd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, jul 2012.

[43] Jonas Diemer and Rolf Ernst. Back suction: Service guarantees for latency-sensitive on-chip networks. In *Proceedings of the 2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip*, NOCS '10, pages 155–162, Washington, DC, USA, 2010. IEEE Computer Society.

[44] Jonas Diemer, Jonas Rox, Mircea Negrean, Steffen Stein, and Rolf Ernst. Real-time communication analysis for networks with two-stage arbitration. In *Proceedings of the 11th International Conference on Embedded Software, EMSOFT 2011, part of the Seventh Embedded Systems Week, ESWeek 2011, Taipei, Taiwan, October 9-14, 2011*, 2011.

[45] Jonas Fabian Diemer. *Predictable Architecture and Performance Analysis for General-Purpose Networks-on-Chip*. PhD thesis, TU Braunschweig, 2016.

[46] Guy Durrieu, Madeleine Faugère, Sylvain Girbal, Daniel Gracia Pérez, Claire Pagetti, and Wolfgang Puffitsch. Predictable flight management system implementation on a multicore processor. In *Embedded Real Time Software and Systems*, ERTS '14, 2014.

[47] Leonardo Ecco and Rolf Ernst. Improved dram timing bounds for real-time dram controllers with read/write bundling. In *2015 IEEE Real-Time Systems Symposium*, pages 53–64, Dec 2015.

[48] Egbert G.T. Jaspers Erik B. van der Tol. Mapping of mpeg-4 decoding on a flexible architecture platform, 2001.

[49] Rolf Ernst. Bringing dynamic control to real-time nocs. In *16th International Forum on MPSoC for Software-defined Hardware*, Nara, Japan, July 11-15 2016.

[50] Rolf Ernst et. al. Report from the workshop on mixed criticality systems. *European Commission, Information Society and Media Directorate-General Unit G3/Computing Systems Research Objective*, 2012.

[51] Yuko Hara et al. Proposal and quantitative analysis of the chstone benchmark program suite for practical c-based high-level synthesis. *Journal of Information Processing*, 17:242–254, 2009.

[52] Mohammad Fattah, Masoud Daneshtalab, Pasi Liljeberg, and Juha Plosila. Smart hill climbing for agile dynamic mapping in many-core systems. In *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, pages 39:1–39:6, New York, NY, USA, 2013. ACM.

[53] Jiri Gaisler, Edvin Catovic, Marko Isomaki, Kristoffer Glembo, and Sandi Habinc. Grlib ip core user's manual. *Gaisler research*, 2007.

[54] Manil Dev Gomony, Benny Akesson, and Kees Goossens. A real-time multi-channel memory controller and optimal mapping of memory clients to memory channels. *ACM Transactions on Embedded Computing Systems (TECS)*, 2014.

[55] K. Goossens and A. Hansson. The aethereal network on chip after ten years: Goals, evolution, lessons, and future. In *Design Automation Conference*, pages 306–311, June 2010.

[56] Kees Goossens, John Dielissen, and Andrei Radulescu. Aethereal network on chip: Concepts, architectures, and implementations. *IEEE Des. Test*, 22(5):414–421, September 2005.

[57] Boris Grot, Joel Hestness, Stephen W. Keckler, and Onur Mutlu. Kilo-noc: A heterogeneous network-on-chip architecture for scalability and service guarantees. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 401–412, New York, NY, USA, 2011. ACM.

[58] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538)*, pages 3–14, Dec 2001.

[59] A. Hansson, M. Coenen, and K. Goossens. Channel trees: Reducing latency by sharing time slots in time-multiplexed networks on chip. In *2007 5th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 149–154, Sept 2007.

[60] John Hauser. *SoftFloat, http://www.jhauser.us/arithmetic/SoftFloat.html*, Accessed online 29.11.2017.

[61] Rafik Henia, Arne Hamann, Marek Jersak, Razvan Racu, Kai Richter, and Rolf Ernst. System level performance analysis - the symta/s approach. *IEE Proceedings Computers and Digital Techniques*, 2005.

[62] John L. Hennessy and David A. Patterson. *Computer Architecture, Fifth Edition: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5th edition, 2011.

[63] Jingcao Hu and R. Marculescu. Energy- and performance-aware mapping for regular noc architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(4):551–562, April 2005.

[64] A. C. Hung. *"PVRG-JPEG CODEC 1.1" Technical Report*, Stanford University, 1993.

[65] L. S. Indrusiak, J. Harbin, and A. Burns. Average and worst-case latency improvements in mixed-criticality wormhole networks-on-chip. In *2015 27th Euromicro Conference on Real-Time Systems*, pages 47–56, July 2015.

[66] Leandro Soares Indrusiak. End-to-end schedulability tests for multiprocessor embedded systems based on networks-on-chip with priority-preemptive arbitration. *Journal of Systems Architecture*, 60(7):553 – 561, 2014.

[67] ISO26262. Road vehicles – Functional safety, 2011.

[68] JEDEC, Arlington, Va, USA. *JESD79-3F: DDR3 SDRAM Specification*, July 2012.

[69] Ahmed Jerraya and Wayne Wolf. *Multiprocessor systems-on-chips*. Elsevier, 2004.

[70] Nan Jiang, Daniel U. Becker, George Michelogiannakis, James D. Balfour, Brian Towles, David E. Shaw, John Kim, and William J. Dally. A detailed and flexible cycle-accurate network-on-chip simulator. In *ISPASS*, pages 86–96. IEEE Computer Society, 2013.

[71] Kalray. *"MPPA Getting Started Guide"*, 2016.

[72] Evangelia Kasapaki and Jens Spars. *The Argo NOC: Combining TDM and GALS*. IEEE, 2015.

[73] J. Kim. Low-cost router microarchitecture for on-chip networks. In *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 255–266, Dec 2009.

[74] John Leslie King. Centralized versus decentralized computing: Organizational considerations and management options. *ACM Comput. Surv.*, 15(4):319–349, December 1983.

[75] A. Kostrzewa, S. Saidi, and R. Ernst. Slack-based resource arbitration for real-time networks-on-chip. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1012–1017, March 2016.

[76] A. Kostrzewa, S. Tobuschat, P. Axer, and R. Ernst. Supervised sharing of virtual channels in networks -on-chip. In *Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems (SIES 2014)*, pages 133–140, June 2014.

[77] A. Kostrzewa, S. Tobuschat, L. Ecco, and R. Ernst. Adaptive load distribution in mixed-critical networks-on-chip. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 732–737, Jan 2017.

[78] A. Kostrzewa, S. Tobuschat, R. Ernst, and S. Saidi. Safe and dynamic traffic rate control for networks-on-chips. In *2016 Tenth IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, pages 1–8, Aug 2016.

[79] Adam Kostrzewa, Rolf Ernst, and Selma Saidi. Multi-path scheduling for multimedia traffic in safety critical on-chip network. In *Proceedings of the 14th ACM/IEEE Symposium on Embedded Systems for Real-Time Multimedia*, ESTIMedia'16, pages 37–46, New York, NY, USA, 2016. ACM.

[80] Adam Kostrzewa, Selma Saidi, Leonardo Ecco, and Rolf Ernst. Flexible tdm-based resource management in on-chip networks. In *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, RTNS '15, pages 151–160, New York, NY, USA, 2015. ACM.

[81] Adam Kostrzewa, Selma Saidi, Leonardo Ecco, and Rolf Ernst. Dynamic admission control for real-time networks-on-chips. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 719–724, Jan 2016.

[82] Adam Kostrzewa, Selma Saidi, Leonardo Ecco, and Rolf Ernst. Ensuring safety and efficiency in networks-on-chip. *Integration, the VLSI Journal*, 58(Supplement C):571 – 582, 2017.

[83] Adam Kostrzewa, Selma Saidi, and Rolf Ernst. Dynamic control for mixed-critical networks-on-chip. In *Proceedings of the 2015 IEEE Real-Time Systems Symposium (RTSS)*, RTSS '15, pages 317–326, San Antonio, TX, USA, 2015. IEEE Computer Society.

[84] Adam Kostrzewa, Sebastian Tobuschat, and Rolf Ernst. Self-aware network-on-chip control in real-time systems. *IEEE Design and Test*, 2018.

[85] Adam Kostrzewa, Sebastian Tobuschat, Selma Saidi, and Rolf Ernst. Supporting suspension-based locking mechanisms for real-time networks-on-chip. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, RTNS '16, pages 215–224, New York, NY, USA, 2016. ACM.

[86] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, Jan 2015.

[87] Daniel Kroening. *SNU, http://www.cprover.org/goto-cc/examples/snu.html*, Accessed online 29.11.2017.

[88] M. Gaur V. Laxmi L. Jain, B. Al-Hashimi and A. Narayanan. Nirgam: A simulator for noc interconnect routing and applications' modeling1. In *Workshop on Diagnostic Services in Network-on-Chips, Design, Automation and Test in Europe Conference (DATE'07)*, page 16–20, 2007.

[89] Jean-Yves Le Boudec and Patrick Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer-Verlag, Berlin, Heidelberg, 2001.

[90] Chunho Lee, M. Potkonjak, and W. H. Mangione-Smith. Mediabench: a tool for evaluating and synthesizing multimedia and communications systems. In *Proceedings of 30th Annual International Symposium on Microarchitecture*, pages 330–335, Dec 1997.

[91] J.P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Real-Time Systems Symposium, 1990. Proceedings., 11th*, pages 201–209, December 1990.

[92] Jane W. S. W. Liu. *Real-Time Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000.

[93] Z. Lu and A. Jantsch. Tdm virtual-circuit configuration for network-on-chip. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 16(8):1021–1034, Aug 2008.

[94] Zhonghai Lu, Rikard Thid, Mikael Millberg, and Axel Jantsch. Nnse: Nostrum network-on-chip simulation environment. In *In Proc. of SSoCC*, 2005.

[95] Amilcar Do Carmo Lucas, Henning Sahlbach, Sean Whitty, Sven Heithecker, and Rolf Ernst. Application development with the flexwafe real-time stream processing architecture for fpgas. *ACM Trans. Embed. Comput. Syst.*, 9(1):4:1–4:23, October 2009.

[96] K. Mahmood, A. Chilwan, O. sterb, and M. Jarschel. Modelling of openflow-based software-defined networks: the multiple node case. *IET Networks*, 4(5):278–284, 2015.

[97] R. Manevich, I. Walter, I. Cidon, and A. Kolodny. Best of both worlds: A bus enhanced noc (benoc). In *2009 3rd ACM/IEEE International Symposium on Networks-on-Chip*, pages 173–182, May 2009.

[98] David Manners. *Auto is fastest growing IC market, says IC Insights.* electronics-weekly.com, (Online on 21.02.2018), 9th November 2017.

[99] R. Marculescu. Worm_sim: a cycle accurate simulator for networks-on-chip. In *Carnegie Mellon University, Website (access 01.11.2017)*, 2017.

[100] R. Marculescu, J. Hu, and U. Y. Ogras. Key research problems in noc design: a holistic perspective. In *2005 Third IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'05)*, pages 69–74, Sept 2005.

[101] N. McKeown. The islip scheduling algorithm for input-queued switches. *IEEE/ACM Transactions on Networking*, 7(2):188–201, Apr 1999.

[102] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008.

[103] Kraig Meyer, Mike Erlinger, Joe Betser, Carl Sunshine, Germán Goldszmidt, and Yechiam Yemini. *Decentralizing Control and Intelligence in Network Management*, pages 4–16. Springer US, Boston, MA, 1995.

[104] Naveen Muralimanohar and Rajeev Balasubramonian. Interconnect design considerations for large nuca caches. *SIGARCH Comput. Archit. News*, 35(2):369–380, June 2007.

[105] Mircea Negrean, Sebastian Klawitter, and Rolf Ernst. Timing analysis of multi-mode applications on AUTOSAR conform multi-core systems. In *Design, Automation and Test in Europe, DATE 13, Grenoble, France, March 18-22, 2013*, pages 302–307, 2013.

[106] Mircea Negrean, Simon Schliecker, and Rolf Ernst. Response-time analysis of arbitrarily activated tasks in multiprocessor systems with shared resources. In *Proc. of Design, Automation, and Test in Europe (DATE)*, Nice, France, apr 2009.

[107] Mircea Negrean, Simon Schliecker, and Rolf Ernst. Timing implications of sharing resources in multicore real-time automotive systems. *SAE International Journal of Passenger Cars - Electronic and Electrical Systems*, 3(1):27–40, aug 2010.

[108] M. Neukirchner, T. Michaels, P. Axer, S. Quinton, and R. Ernst. Monitoring arbitrary activation patterns in real-time systems. In *2012 IEEE 33rd Real-Time Systems Symposium*, pages 293–302, Dec 2012.

[109] AMPG Body Electronics Systems Engineering Team NXP. Future advances in body electronics. 2015.

[110] Umit Y. Ogras, Jingcao Hu, and Radu Marculescu. Key research problems in noc design: A holistic perspective. In *Proceedings of the 3rd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES+ISSS '05, pages 69–74, New York, NY, USA, 2005. ACM.

[111] Omnest. *https://www.omnest.com/publications.php*, Accessed online 1.12.2017.

[112] OSEK Group. OSEK/VDX Operating System Specification, 2015.

[113] R. Pellizzoni, E. Betti, S. Bak, G. Yao, J. Criswell, M. Caccamo, and R. Kegley. A predictable execution model for cots-based embedded systems. In *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 269–279, April 2011.

[114] Quentin Perret, Pascal Maurère, Éric Noulard, Claire Pagetti, Pascal Sainrat, and Benoît Triquet. Mapping hard real-time applications on many-core processors. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, RTNS '16, pages 235–244, New York, NY, USA, 2016. ACM.

[115] D. Pham, S. Asano, M. Bolliger, M. N. Day, H. P. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Riley, D. Shippy, D. Stasiak, M. Suzuoki, M. Wang, J. Warnock, S. Weitzel, D. Wendel, T. Yamazaki, and K. Yazawa. The design and implementation of a first-generation cell processor. In *ISSCC. 2005 IEEE International Digest of Technical Papers. Solid-State Circuits Conference, 2005.*, pages 184–592 Vol. 1, Feb 2005.

[116] A. Psarras, I. Seitanidis, C. Nicopoulos, and G. Dimitrakopoulos. Phasenoc: Tdm scheduling at the virtual-channel level for efficient network traffic isolation. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1090–1095, March 2015.

[117] V. Puente, J. A. Gregorio, and R. Beivide. Sicosys: an integrated framework for studying interconnection network performance in multiprocessor systems. In *Proceedings 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*, pages 15–22, 2002.

[118] S. Quinton, T. T. Bone, J. Hennig, M. Neukirchner, M. Negrean, and R. Ernst. Typical worst case response-time analysis and its use in automotive network design. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2014.

[119] Rafik Henia Arne Hamann Rolf Ernst Razvan Racu, Li Li. Improved response time analysis of tasks scheduled under preemptive round-robin. In *Proc. of the International Conference on Hardware-Software Codesign and System Synthesis*, pages 179–184, Salzburg, Austria, oct 2007.

[120] Renesans. *R-Car H3 Architecture Overview*. renesas.com, (Online on 21.02.2018), 2017.

[121] Kai Richter. *Compositional Scheduling Analysis Using Standard Event Models*. PhD thesis, TU Braunschweig, IDA, 2005.

[122] E. Rijpkema, K. G. W. Goossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander. Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip. In *2003 Design, Automation and Test in Europe Conference and Exhibition*, pages 350–355, 2003.

[123] Scott Rixner, William J. Dally, Ujval J. Kapasi, Peter Mattson, and John D. Owens. Memory access scheduling. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, ISCA '00, pages 128–138, New York, NY, USA, 2000. ACM.

[124] R. Sandoval-Arechiga, R. Parra-Michel, J. L. Vazquez-Avila, J. Flores-Troncoso, and S. Ibarra-Delgado. Software defined networks-on-chip for multi/many-core systems: A performance evaluation. In *2016 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pages 129–130, March 2016.

[125] L. Schenato, B. Sinopoli, M. Franceschetti, K. Poolla, and S.S. Sastry. Foundations of control and estimation over lossy networks. *Proceedings of the IEEE*, 95(1):163–187, January 2007.

[126] Bastian Schlich. Model checking of software for microcontrollers. *ACM Trans. Embed. Comput. Syst.*, 9(4):36:1–36:27, April 2010.

[127] Simon Schliecker, Mircea Negrean, and Rolf Ernst. Response time analysis in multicore ecus with shared resources. *IEEE Transactions on Industrial Informatics*, 5(4), nov 2009.

[128] Simon Schliecker, Jonas Rox, Matthias Ivers, and Rolf Ernst. Providing accurate event models for the analysis of heterogeneous multiprocessor systems. In *Proceedings of the 6th IEEE/ACM/IFIP International Conference on Hardware/-Software Codesign and System Synthesis*, CODES+ISSS '08, pages 185–190, New York, NY, USA, 2008. ACM.

[129] Martin Schoeberl, Florian Brandner, Jens Spars, and Evangelia Kasapaki. A statically scheduled time-division-multiplexed network-on-chip for real-time systems. In *Proceedings of the 2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, NOCS '12, pages 152–160, Washington, DC, USA, 2012. IEEE Computer Society.

[130] Zheng Shi and Alan Burns. Real-time communication analysis for on-chip networks with wormhole switching. In *Proceedings of the Second ACM/IEEE International Symposium on Networks-on-Chip*, NOCS '08, pages 161–170, Washington, DC, USA, 2008. IEEE Computer Society.

[131] David Spieker. *Dynamische Verwaltung von virtuellen Kanälen im IDA NoC Netzwerk Interface.* Bachelorarbeit, TU Braunschweig, März 2015.

[132] John A. Stankovic, Krithi Ramamritham, and Marco Spuri. *Deadline Scheduling for Real-Time Systems: Edf and Related Algorithms.* Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[133] R. Stefan, A. Molnos, A. Ambrose, and K. Goossens. A tdm noc supporting qos, multicast, and fast connection set-up. In *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1283–1288, March 2012.

[134] R. A. Stefan, A. Molnos, and K. Goossens. daelite: A tdm noc supporting qos, multicast, and fast connection set-up. *IEEE Transactions on Computers*, 63(3):583–594, March 2014.

[135] D. Stiliadis and A. Varma. Latency-rate servers: a general model for analysis of traffic scheduling algorithms. *IEEE/ACM Transactions on Networking*, 6(5):611–624, Oct 1998.

[136] Andrew Tanenbaum. *Computer Networks*. Prentice Hall Professional Technical Reference, 5th edition, 2011.

[137] Daniel Thiele and Rolf Ernst. Formal analysis based evaluation of software defined networking for time-sensitive ethernet. In *Design Automation and Test in Europe (DATE)*, Dresden, Germany, March 2016.

[138] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *2000 IEEE International Symposium on Circuits and Systems. Emerging Technologies for the 21st Century. Proceedings (IEEE Cat No.00CH36353)*, volume 4, pages 101–104 vol.4, 2000.

[139] Ken W Tindell, Alan Burns, and Andy J Wellings. An extendible approach for analyzing fixed priority hard real-time tasks. *Real-Time Systems*, 6(2):133–151, 1994.

[140] S. Tobuschat, P. Axer, R. Ernst, and J. Diemer. Idamc: A noc for mixed criticality systems. In *2013 IEEE 19th International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 149–156, Aug 2013.

[141] S. Tobuschat and R. Ernst. Efficient latency guarantees for mixed-criticality networks-on-chip. In *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 113–122, April 2017.

[142] S. Tobuschat and R. Ernst. Real-time communication analysis for networks-on-chip with backpressure. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 590–595, March 2017.

[143] S. Tobuschat, R. Ernst, A. Hamann, and D. Ziegenbein. System-level timing feasibility test for cyber-physical automotive systems. In *2016 11th IEEE Symposium on Industrial Embedded Systems (SIES)*, pages 1–10, May 2016.

[144] S. Tobuschat, M. Neukirchner, L. Ecco, and R. Ernst. Workload-aware shaping of shared resource accesses in mixed-criticality systems. In *Hardware/-Software Codesign and System Synthesis (CODES+ISSS), 2014 International Conference on*, pages 1–10, October 2014.

[145] Sebastian Tobuschat and Rolf Ernst. Providing throughput guarantees in mixed-criticality Networks-on-Chip. In *2017 30th IEEE International System-on-Chip Conference (SOCC) (SOCC 2017)*, pages 207–212, Munich, Germany, September 2017.

[146] Sebastian Tobuschat, Adam Kostrzewa, and Rolf Ernst. Selective congestion control for mixed-critical networks-on-chip. 12 2017.

[147] I. Walter, I. Cidon, R. Ginosar, and A. Kolodny. Access regulation to hot-modules in wormhole nocs. In *First International Symposium on Networks-on-Chip (NOCS'07)*, pages 137–148, May 2007.

[148] E. Wandeler and L. Thiele. Interface-based design of real-time systems with hierarchical scheduling. In *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*, pages 243–252, April 2006.

[149] Zichen Wang. Evaluation, implementation, integration and testing of the control layer based resource management strategy for a performance optimized network-on-chip. Master's thesis, TU Braunschweig, Januar 2017.

[150] Hassan M. G. Wassel, Ying Gao, Jason K. Oberg, Ted Huffmire, Ryan Kastner, Frederic T. Chong, and Timothy Sherwood. Surfnoc: A low latency and provably non-interfering approach to secure networks-on-chip. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pages 583–594, New York, NY, USA, 2013. ACM.

[151] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C. C. Miao, J. F. Brown III, and A. Agarwal. On-chip interconnection architecture of the tile processor. *IEEE Micro*, 27(5):15–31, Sept 2007.

[152] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3):36:1–36:53, May 2008.

[153] Z. P. Wu, Y. Krish, and R. Pellizzoni. Worst case analysis of dram latency in multi-requestor systems. In *2013 IEEE 34th Real-Time Systems Symposium*, pages 372–383, Dec 2013.

[154] Bing Xiong, Kun Yang, Jinyuan Zhao, Wei Li, and Keqin Li. Performance evaluation of openflow-based software-defined networks based on queueing model. *Comput. Netw.*, 102(C):172–185, June 2016.

[155] Hui Zhang. Service disciplines for guaranteed performance service in packet-switching networks. *Proceedings of the IEEE*, 83(10):1374–1396, Oct 1995.

[156] Dirk Ziegenbein and Arne Hamann. Timing-aware control software design for automotive systems. In *Proceedings of the 52nd Annual Design Automation Conference, San Francisco, CA, USA, June 7-11, 2015*, pages 56:1–56:6, 2015.

Acronyms

ADAS Advanced Driver Assistance System. 155

BC Burst Count. 136

BE best-effort. 12, 14, 86, 102, 103, 111, 118–120

BI Bank Interleaving. 136

CPU central processing unit. 7

DSP digital signal processor. 7

ECU Electronic Control Unit. 10

EDA Electronic Design Automation. 87

FAA Federal Aviation Administration. 15

GALS globally asynchronous locally synchronous. 7

GS Guaranteed Service. 36

HRT hard real-time. 12, 13, 15

HRTT hard real-time transmission. 13–15

HW hardware. 85, 87

IEC International Electrotechnical Commision. 16

IP intellectual property. 7

LUT Look-UP Table. 124

MMU Memory Management Unit. 49

MPSoC Multiprocessor System-on-Chip. 8, 11, 85

MPU Memory Protection Unit. 49

MSoC Manycore System-on-Chip. 11

NI Network Interface. 8, 85–87, 104, 106, 113, 117, 118, 123–125, 127

NIU Network Interface Unit. 8

NoC Network-on-Chip. 5, 8, 10, 12, 14, 16, 24, 25, 27, 46, 85–94, 96–102, 104–112, 114, 115, 117–120, 122–124, 126, 127

OS operating system. 87

QoS quality-of-service. 14, 85

RL Rate Limiter. 39

RM Ressource Manager. 63

RT Real-Time. 35, 36

RTE Run-Time Environment. 87

RTL register-transfer level. 103

SDN Software-defined Networking. 53, 87

SoC System-on-Chip. 7, 11, 12, 14, 15, 20

SRT soft real-time. 12, 13, 15

SRTT soft real-time transimssion. 13–15

SW software. 87

TDM Time-Divsion Multiplexing. 33

TLM transaction-level modeling. 103, 104

VC Virtual Channel. 36

VLSI very-large-scale integration. 7

Glossary

admission control A validation process performed at runtime before a communication is established to see if the currently available NoC resources are sufficient for the particular transmission.. 56

backpressure "Information about the utilization of downstream resources. Backpressure information is used by flow control to prevent overflow of buffers and can be used by an adaptive routing algorithm to avoid congested resources, for example." from [37].. 18

connection The contract-based end-to-end resource reservations for a given sender-receiver pair on the selected path with a selected QoS provisioning.. 57

criticality Criticality is a designation of the level of assurance against failure needed for a system component from [30].. 16

data flow "The directed transmission of data between multiple entities", from [36].. 65

deadline "The point in time when an execution of an entity must be finished", from [36].. 12

event "State change of a hardware and/or software entity", from [36].. 18

fail-operational Feature of a system or a unit. Describes the ability of a system or functional unit to continue normal operation despite the presence of hardware or software faults, based on [36].. 21

fail-safe Feature of a system or a unit. In case of a fault the system or the functional unit transits to a safe state, based on [36].. 21

failure Termination of the ability of a system of functional unit to perform a required function, from [36].. 18

fault abnormal condition that can cause an element or an item to fail. 18

flow control "Flow control is the scheduling and allocation of a network's resources, such as channel bandwidth, buffer space, and control state." from [37].. 18

global state of the system The number of currently running applications and their current requirements with respect to the shared system resources e.g. NoC's links and buffers.. 57

hot-module A hot-spot module is a module whose demand is significantly greater than other, similar resources. For example, a particular destination terminal becomes a hot-spot in a shared memory multicomputer when many processing nodes are simultaneously reading from the node, based on [37].. 144

interfering transmissions Transmissions which overlap in at least one router on their path from source to destination and therefore are sharing NoC resources, i.e., link bandwidth and/or buffers in routers.. 56

jitter "The maximum difference in the latency between two packets within a flow. Low jitter is often a requirement for video streams or other real time data for which the regularity of data arrival is important. The jitter times the bandwidth of a flow gives a lower bound on the size of buffer required." from [37].. 23

latency time required to deliver a unit of data (usually a packet or mes- sage) through the network, measured from the the injec- tion of the first bit at the source to the ejection of the last bit at the destination [37]. 12

mixed-criticality system A system where applications of different levels of criticality are executed on a shared computing platform from [30].. 16

non-blocking "A network is non-blocking if it can simultaneously handle all circuit requests that are a permutation of the inputs and outputs. A non-blocking network can always handle a request for a circuit from any idle input to any idle output.", from [37].. 72

predictability accuracy of the formal evaluation (i.e. calculation or ”prediction”) of the observed run-time behavior for the system or system's element (e.g. latency of transmissions for interconnect) without (full) knowledge of run-time workloads (e.g. deployed traffic, senders behavior), cf.[45].. 85

QoS provisioning The Quality-of-Service (QoS) provisioning defines the minimal set of resources requested for connections (reservations) including at least the type and desired upper time response of the NoC depending on the particular synchronization scenario.. 57

quality of service The bandwidth, latency, and/or jitter received by a particular connection or class of traffic. A QoS policy differentiates between connections and provides services to those connections based on a contract that guarantees the QoS provided to each connection, provided that the connection complies with restrictions on volume and burstiness of traffic, based on [37].. 29

real-time systems The system in which the correctness of the system design and working depends equally on temporal and functional aspects.. 12

response time "The length of time from the release time of the job to the instant when it completes.", from [92].. 57

risk Combination of the probability of occurrence of harm and the severity, from [36].. 16

safety "Freedom from unacceptable risk to persons and goods", from [36].. 15

scalability "The degree to which assets (e.g. NoC) can be adapted to specific target environments for various defined measures.", from [36].. 20

synchronization scenarios Sets of applications which may mutually influence their execution times, e.g., through concurrent accesses to shared interconnect resources.. 56

throughput "The amount of traffic (in bits/s) delivered to the destination terminals of the network", from [37].. 12

timeout Notification with respect to deadline violation of an event or task (e.g. while working on/with information: receiving, sending, processing), from [36].. 65

topology The static arrangement of routers, links, and processing nodes in a network, based on [37].. 3

traffic locality Feature of the NoC assuring that packets belonging to the same logical connection arrive in the same specific order in which they were injected to the NoC (in-order delivery) and / or that streams from two different senders targeting the same node do not mix in the NoC.. 23

use-case "A model of the usage by the user of a system in order to realize a certain functional feature of the system.", from [36].. 117

Adam Kostrzewa
Hans-Sommer-Str. 66
38106 Braunschweig